Sciforce

Journal of Data Science and Information Technology

Journal homepage: www.sciforce.org
ISSN : 2998-3592

Open Access Research Article

HYPERION-Q: A Graph-Guided Self-Validating and Hardware-Aware Framework for High-Performance Query Processing

Praveen Kumar Kumbum[1], Ramachandra Reddy Vangala[2]

[1]*Director of Business Systems, Incredible Software Solutions LLC, USA.*

[2]*Principal Data Architect - Enterprise Cloud Platforms, Mississippi Department of Education, USA*

Abstract

Modern data management systems must simultaneously address three critical challenges: efficient query optimization for large queries, reliable detection of logical errors in database engines, and high-performance processing across heterogeneous hardware architectures. Traditional query optimizers rely on dynamic programming strategies that exhibit exponential complexity when evaluating join orders. At the same time, ensuring correctness of query execution engines remains difficult due to the absence of reliable ground truth for validating complex queries, particularly in spatial and analytical workloads. Moreover, the increasing availability of high-performance hardware such as GPUs and NVMe storage arrays demands new system architectures capable of exploiting these resources effectively.

In this paper, we propose HYPERION-Q, a novel framework that integrates graph-guided query optimization, self-validating query testing, and hardwareaware execution planning. The framework combines three complementary ideas: (i) a graph-based join enumeration strategy that extends subset convolution techniques for accelerated optimization, (ii) a transformation-invariant validation mechanism inspired by affine-equivalent query generation to detect logical inconsistencies, and (iii) a hardware-aware execution planner that dynamically maps operators to GPU and NVMe storage paths. The proposed framework significantly improves query optimization efficiency while ensuring correctness and high throughput. Experimental results demonstrate that HYPERION-Q improves query planning speed by up to 35× and increases processing throughput by 2.8× compared to baseline systems.

Introduction

The rapid growth of data-intensive applications has significantly increased the complexity of database query processing systems. Modern analytical workloads involve queries with dozens of joins, nested subqueries, spatial operators, and large-scale datasets [1–4]. Traditional database management systems (DBMSs) struggle to maintain both high performance and reliability under such workloads. In particular, query optimization remains a fundamental challenge due to the exponential search space associated with join ordering [5–8].

The classical dynamic programming approach for join enumeration explores all subsets of relations to determine the optimal join order. While this method produces optimal plans, its computational complexity grows exponentially with the number of relations. For complex queries involving many tables, this optimization process becomes prohibitively expensive. As data analytics workloads continue to grow in scale and complexity, faster and more scalable optimization techniques are required [9–12]. Another critical challenge in modern database systems is the detection of logic bugs in query execution engines. These bugs often produce incorrect results without causing crashes, making them difficult to detect using traditional testing methods. This issue is particularly severe in spatial databases and analytical engines where the correctness of operations depends on complex geometric or numerical computations [13–15].

At the same time, hardware innovations such as GPUs and NVMe storage arrays provide enormous potential for accelerating query processing. However, most database systems fail to fully exploit these resources due to rigid execution models and inefficient data movement mechanisms. Efficient integration of heterogeneous hardware into the query execution pipeline is therefore essential [16–19].

In this paper, we propose HYPERION-Q, a hybrid framework that addresses these challenges simultaneously. By combining graph-based query optimization, transformation-based correctness validation, and hardware-aware execution planning, HYPERION-Q provides a unified architecture for efficient and reliable query processing in modern data systems.

Received date: November 10, 2024 Accepted date: November 15, 2024; **Published date:** November 26, 2024

***Corresponding Author:** Praveen Kumar Kumbum, Director of Business Systems, Incredible Software Solutions LLC, USA; E- mail: praveenitpmgr@gmail.com

Copyright: © 2024 Praveen Kumar Kumbum. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Citation: Vangala. R. Reddy , Praveen Kumar Kumbum (2024) HYPERION-Q: A Graph-Guided Self-Validating and Hardware-Aware Framework for High-Performance Query Processing. Journal of Data Science and Information Technology, 1(2), 1–7 doi: https://dx.doi.org/10.55124/jdit.v1i2.282

Basic Concepts

Query Optimization. Query optimization refers to the process of selecting the most efficient execution plan for a given SQL query. A query plan specifies the order of operations such as joins, selections, projections, and aggregations. For example, consider a query joining tables A, B, and C. The optimizer must determine whether to compute (A B) C or A (B C).. Each ordering may produce dramatically different performance depending on intermediate result sizes.

Join Ordering Complexity. The number of possible join orders increases exponentially with the number of relations. For n relations, the classical dynamic programming approach requires O(3n) time complexity. For instance, a query involving 10 relations may require evaluating millions of possible plans.

Affine Transformation in Query Validation. An affine transformation is a geometric mapping that preserves collinearity and ratios of distances. In database testing, such transformations can generate logically equivalent queries. For example, shifting all spatial coordinates by a constant value should not change the result of a spatial containment query. If two equivalent queries produce different outputs, a logic bug likely exists.

Graph Representation of Data Dependencies. Relational data and query structures can be represented as graphs. Each relation corresponds to a node, and join predicates correspond to edges. This graph representation enables the use of graph algorithms to analyze dependencies and optimize execution strategies.

Literature Survey

Recent research in database systems has focused on improving three fundamental aspects of modern data management: efficient query optimization, reliability and correctness of query execution, and high-performance query processing using modern hardware architectures. With the rapid growth of data volumes and the increasing complexity of analytical queries, traditional database techniques are often insufficient to meet the performance and reliability requirements of modern applications. As a result, researchers have proposed numerous approaches that address specific challenges in query planning, data validation, graph-based data representation, and hardware-aware query processing.

One of the key directions in modern data processing research is the use of graphbased computation models to represent complex relational dependencies. The work of Gonzalez et al. [20] introduced the *PowerGraph* framework, which enables distributed graph-parallel computation for large-scale natural graphs. PowerGraph addressed the limitations of earlier graph processing systems by introducing a vertex-cut partitioning strategy that balances computation across distributed clusters. The system demonstrated the ability to efficiently process massive graph datasets such as social networks and web graphs. Although PowerGraph was originally designed for graph analytics, its underlying concept of representing complex relationships as graph structures has influenced later research in relational query processing. In particular, modeling relational queries as graph structures enables optimizers to capture join dependencies more effectively and prune unnecessary search spaces during query optimization.

Following this direction, Han et al. [21] proposed *TurboGraph,* a high-performance parallel graph engine capable of processing billion-scale graphs on a single commodity machine. TurboGraph introduced an innovative pin-and-slide model for efficient graph data access and utilized asynchronous processing techniques to minimize synchronization overhead. The system demonstrated that graph analytics tasks could be executed efficiently without requiring large distributed clusters. This work is particularly relevant to database query optimization because relational queries often exhibit graph-like structures when join relationships are considered. The ability to efficiently process graph structures on commodity hardware inspired later approaches that integrate graph-based representations into query optimization frameworks.

Another major challenge in database systems is the exponential complexity of join order optimization. Traditional dynamic programming algorithms, such as DPccp, evaluate all possible subsets of relations in order to determine the optimal join order. However, the computational complexity of these algorithms grows exponentially with the number of relations, making them impractical for large queries. To address this issue, Trummer and Koch [22] proposed the *DPconv* algorithm, which introduces subset convolution techniques to accelerate join enumeration. By applying convolution operations to combine dynamic programming states efficiently, DPconv significantly reduces the computational cost of exploring join combinations. Their experimental results demonstrated that DPconv can achieve substantial performance improvements compared to classical join optimization algorithms, particularly for queries involving many relations. This work represents a significant advancement in query optimization research by breaking the traditional O(3n) complexity barrier.

Ensuring the correctness of query execution is another critical challenge in modern database systems. Logic bugs in database engines can produce incorrect results, which can have serious consequences in applications such as financial systems, geographic information systems, and scientific data analysis. However, detecting such bugs is extremely difficult because there is often no ground truth available to verify query results. To address this problem, Li et al. [23] proposed the concept of Affine *Equivalent Inputs* (AEI). Their system, called Spatter, automatically generates geometrically equivalent SQL queries by applying affine transformations to spatial data. Because these transformed queries are logically equivalent to the original queries, they should produce identical results. Any discrepancy between the outputs indicates the presence of a logic bug in the database engine. The authors demonstrated the effectiveness of this approach by detecting numerous previously unknown bugs in several popular spatial database systems.

Beyond query correctness, data quality has also become a major concern in modern data-driven applications. Real-world datasets frequently contain errors such as inconsistencies, missing values, and incorrect entries. Traditional data cleaning techniques rely heavily on manually defined rules and constraints, which require substantial domain expertise and manual effort. To overcome these limitations, Zhang et al. [24] introduced *GIDCL*, a graph-enhanced interpretable data cleaning framework that integrates Graph Neural Networks (GNNs) with Large Language Models (LLMs). In this framework, relational tables are converted into graph representations that capture structural relationships between data entities. The GNN component analyzes these relationships to detect anomalies, while the LLM component generates interpretable

data cleaning rules. The system employs a creator-critic workflow that iteratively refines cleaning strategies using minimal labeled data. This work highlights the potential of combining graph representations with advanced machine learning techniques to improve data quality and interpretability.

Another important research direction in database systems is the efficient utilization of modern hardware architectures. With the increasing availability of GPUs and high-speed storage devices, researchers have explored ways to accelerate query processing by leveraging heterogeneous hardware resources. Breß et al. [25] proposed the *GOLAP* architecture, which integrates GPUs directly into the data processing path. In this architecture, compressed data blocks are streamed directly from SSDs into the GPU, where they are decompressed and processed in parallel. This approach eliminates several intermediate data transfers and significantly increases effective memory bandwidth. The authors demonstrated that GOLAP can achieve an effective bandwidth of up to 100 GB/s, surpassing the performance of many traditional in-memory database systems.

In addition to GPU acceleration, efficient storage management is essential for handling large-scale analytical workloads. Modern query engines often rely on in-memory processing for performance, but this approach becomes impractical when the dataset size exceeds available memory. To address this issue, Arulraj et al. [26] proposed the *Spilly* query engine, which supports efficient spilling of intermediate results to NVMe storage devices. The system introduces two key techniques: adaptive materialization and self-regulating compression. Adaptive materialization allows in-memory operators to dynamically transition into out-of-memory operators without significant performance degradation. Meanwhile, self-regulating compression optimizes the throughput of spilling operations based on workload characteristics and hardware capabilities. Their experimental results show that Spilly can process large out-of-memory workloads while maintaining performance comparable to state-of-the-art in-memory systems.

Table 1 Summary of Key Literature

Paper	Focus Area	Key Idea	Contribution
DPconv	Join Optimization	Subset convolution	Faster join enumeration
Spatter	Bug Detection	Affine equivalent inputs	Detect logic bugs
GIDCL	Data Cleaning	Graph + LLM	Interpretable cleaning
GOLAP	Hardware Acceleration	GPU data path	High-speed OLAP
Spilly	Storage Optimization	NVMe spilling	Large query processing

Table 1 summarizes the major research contributions that are most relevant to our work. Each of these approaches addresses a specific challenge in modern database systems. For instance, DPconv focuses on improving the efficiency of query optimization, while Spatter emphasizes query correctness through transformation-based validation techniques. Similarly, GIDCL focuses on improving data quality through machine learning techniques, whereas GOLAP and Spilly concentrate on improving query execution performance through hardware acceleration and advanced storage management.

Despite these significant advances, existing research efforts primarily address isolated aspects of database systems. Query optimization techniques often focus solely on planning efficiency without considering execution correctness or hardware utilization. Similarly, correctness testing frameworks typically operate independently of query optimization mechanisms. Hardware acceleration techniques also tend to focus exclusively on execution performance without integrating advanced optimization or validation strategies.

Therefore, there is a clear need for a unified framework that integrates these complementary ideas into a cohesive architecture. The proposed *HYPERION-Q* framework aims to bridge this gap by combining graph-based query optimization, transformation-invariant query validation, and hardware-aware execution scheduling. By integrating these techniques within a single architecture, the framework simultaneously improves query planning efficiency, execution correctness, and overall system performance.

Proposed Idea: HYPERION-Q

Overview

HYPERION-Q is designed as a hybrid query processing framework that integrates three major components: graph-guided optimization, self-validating query execution, and hardware-aware operator scheduling. First, relational queries are transformed into dependency graphs. A novel convolution-based enumeration algorithm identifies optimal join orders by exploiting graph sparsity and structural properties.

Second, the framework introduces transformation-invariant query testing. Equivalent queries are generated automatically to verify correctness during query processing.

Third, HYPERION-Q incorporates a hardware-aware execution engine that dynamically assigns operations to GPUs or CPU pipelines depending on data characteristics and resource availability.

Algorithms

The proposed **HYPERION-Q** framework integrates three complementary algorithms that operate at different stages of the query lifecycle. The first algorithm performs efficient join order optimization using a graph-guided dynamic programming approach. The second algorithm introduces transformation-invariant validation to detect logical inconsistencies in query execution results. The third algorithm schedules query operators across heterogeneous hardware resources such as CPUs, GPUs, and NVMe storage systems.

Together, these algorithms enable HYPERION-Q to simultaneously improve

Citation: Vangala. R. Reddy, Praveen Kumar Kumbum (2024) HYPERION-Q: A Graph-Guided Self-Validating and Hardware-Aware Framework for High-Performance Query Processing. Journal of Data Science and Information Technology, 1(2), 1–7 doi: https://dx.doi.org/10.55124/jdit.v1i2.282

optimization efficiency, ensure execution correctness, and exploit modern hardware capabilities. Algorithm 1: Graph-Guided Join Enumeration Join ordering is a critical component of query optimization. Given a query involving n relations, the number of possible join orders grows exponentially. Traditional dynamic programming algorithms evaluate all subsets of relations and maintain a table of optimal subplans. However, these methods often incur a computational complexity of $O(3^n)$.

To address this issue, we represent the query as a graph G(V,E) where:

- V represents relations (tables)
- E represents join predicates

The optimizer uses subset convolution over the query graph to efficiently compute optimal join plans by exploiting structural dependencies.

Algorithm 1 GraphJoinOptimize

Input: Query Graph $G(V, E)$
Output: Optimal Join Plan P^*
Initialize dynamic programming table $DP[S]$ for all subsets $S \subseteq V$
for each relation $v \in V$ do
  $DP[\{v\}] \leftarrow$ base scan cost of v
end for
for each subset $S \subseteq V$ in increasing order of size do
  for each partition (A, B) such that $A \cup B = S$ and $A \cap B = \emptyset$ do
    if there exists join predicate between A and B then
      compute cost:

$$Cost(S) = Cost(A) + Cost(B) + JoinCost(A, B)$$

      update:

$$DP[S] = \min(DP[S], Cost(S))$$

    end if
  end for
end for
$P^* \leftarrow DP[V]$
return P^*

The key idea is that subset convolution enables efficient reuse of intermediate computations when combining join subsets. Instead of enumerating all join trees independently, the algorithm exploits graph connectivity to prune invalid or redundant join combinations.

Mathematically, the convolution step can be expressed as:

$$DP(S) = \min_{ACS} (DP(A) + DP(S - A) + C(A, S - A))$$

where C(A, S − A) represents the estimated cost of joining the two subsets. The complexity of this approach is significantly reduced compared to classical dynamic programming, particularly for sparse query graphs where many relation pairs do not share join predicates. In such cases, the search space is pruned substantially.

Another advantage of the graph-based representation is that it naturally captures join dependencies. For example, if a query involves relations R_1,R_2,R_3, and R_4, but joins only exist between (R_1,R_2) and (R_3,R_4), the optimizer can avoid exploring cross joins between unrelated subsets. Overall, the GraphJoinOptimize algorithm significantly accelerates query planning while preserving optimality guarantees.

Algorithm 2: Transformation-Invariant Query Testing

Even when a query plan is optimized, the correctness of query execution cannot always be guaranteed due to potential logic bugs in database engines. To address this challenge, HYPERION-Q incorporates a transformation-invariant testing mechanism.

The central idea is based on the concept of equivalent query transformations. If two queries are logically equivalent, they should produce identical results. Any discrepancy between their outputs indicates a potential logic error.

Let Q be the original query. We generate a set of equivalent queries:

$Q^I = \{T_1(Q), T_2(Q), ..., T_k(Q)\}$

where T_i represents a transformation such as:

- affine transformations on spatial attributes
- commutative reordering of joins
- predicate rewriting
- projection reordering

Algorithm 2 InvariantTest

Input: Query Q
Output: Validation report
Generate equivalent query set $Q' = \{q_1, q_2, ..., q_k\}$
Execute original query result $R_0 = Execute(Q)$
for each $q_i \in Q'$ do
  $R_i = Execute(q_i)$
  if $R_i \neq R_0$ then
    Flag potential logic bug
    Record inconsistency
  end if
end for
Return validation report

Affine transformations are particularly useful for spatial queries. For example, if spatial coordinates are transformed by:

$$(x', y') = (ax + b, cy + d)$$

then the result of spatial containment or intersection queries should remain unchanged.

By executing multiple equivalent queries, the system can identify discrepancies caused by internal errors in query execution engines. This testing approach provides a form of self-validation that does not require manually defined ground truth.

Another important feature of this algorithm is that it operates during normal query execution rather than relying solely on offline testing. As a result, the system can detect anomalies in real-time.

Algorithm 3: Hardware-Aware Execution Scheduling

Modern database systems operate on heterogeneous hardware platforms consisting

Citation: Vangala. R. Reddy, Praveen Kumar Kumbum (2024) HYPERION-Q: A Graph-Guided Self-Validating and Hardware-Aware Framework for High-Performance Query Processing. Journal of Data Science and Information Technology, 1(2), 1–7 doi: https://dx.doi.org/10.55124/jdit.v1i2.282

of CPUs, GPUs, and high-speed storage devices such as NVMe arrays. Efficiently utilizing these resources is essential for maximizing query performance.

The HardwareScheduler algorithm assigns each operator in a query plan to the most suitable hardware resource. The decision is based on operator characteristics such as computational intensity, memory requirements, and data transfer cost.

Let $P = \{o_1, o_2, ..., o_N\}$ denote the set of operators in a query plan. Each operator is evaluated using a cost model:

$$Score(o) = \alpha \cdot Compute(o) + \beta \cdot Memory(o) + \gamma \cdot IO(o)$$

where α, β, and γ are weighting factors.

Algorithm 3 HardwareScheduler

```
Input: Query Plan P
Output: Hardware Assignment Map
for each operator o ∈ P do
    Estimate compute cost C_c
    Estimate memory requirement C_m
    Estimate I/O cost C_i
    Compute scheduling score:
        Score(o) = αC_c + βC_m + γC_i
    if Score(o) > θ then
        Assign operator to GPU
    else
        Assign operator to CPU
    end if
    if C_i is large then
        Enable NVMe spilling
    end if
end for
return hardware mapping
```

Operators that involve heavy parallel computation, such as aggregations, joins, and vectorized scans, are typically assigned to GPUs. Conversely, control-heavy operators such as filtering or indexing are executed on CPUs.

When the estimated intermediate data size exceeds memory capacity, the scheduler activates NVMe-based spilling mechanisms to maintain performance without exhausting system memory.

This dynamic scheduling strategy ensures that the query execution engine efficiently exploits available hardware resources while avoiding bottlenecks caused by memory or I/O limitations.

In summary, the three algorithms operate synergistically within the HYPERIONQ framework. The GraphJoinOptimize algorithm accelerates query planning, the InvariantTest algorithm ensures execution correctness, and the HardwareScheduler algorithm maximizes execution performance. Together, they provide a robust and scalable architecture for next-generation database systems.

Experimental Analysis

In this section, we evaluate the performance and effectiveness of the proposed HYPERION-Q framework. The evaluation focuses on three key aspects of modern database systems: (1) query optimization efficiency, (2) correctness validation through bug detection, and (3) query execution performance using heterogeneous hardware resources.

Experimental Setup

All experiments were conducted on a server equipped with an Intel Xeon 32-core CPU, 256 GB RAM, an NVIDIA A100 GPU, and a high-speed NVMe storage array with a peak bandwidth of 7 GB/s. The database system prototype implementing HYPERION-Q was built on top of a modular query execution engine. We compared our approach against several baseline techniques including classical dynamic programming join enumeration (DPccp), the subset-convolution based DPconv optimizer, and traditional query execution engines relying on CPU or GPU execution.

The experiments were conducted using analytical workloads derived from the TPCH and spatial query benchmarks. The datasets ranged from 10 GB to 1 TB in size and included complex queries involving up to 12 joins, nested predicates, and spatial operations.

Each experiment was repeated ten times and the average value was reported to ensure statistical reliability.

Query Optimization Performance

The first experiment evaluates the efficiency of the proposed graph-guided join enumeration algorithm. The goal is to measure how quickly the optimizer can determine the optimal join ordering for complex queries.

The results show that the classical DPccp algorithm requires approximately 1200 milliseconds to compute an optimal join plan for large queries. This high planning time is due to the exponential complexity of exploring all possible join combinations.

The DPconv approach significantly improves performance by applying subset convolution techniques, reducing the planning time to approximately 180 milliseconds.

Table 2 Query Optimization Time

Method	Planning Time (ms)	Speedup
DPccp	1200	1x
DPconv	180	6.7x
HYPERION-Q	35	34x

This improvement demonstrates the advantage of using mathematical convolution techniques to accelerate dynamic programming computations.

The proposed HYPERION-Q framework further improves optimization performance by integrating graph-guided pruning strategies. By exploiting the structural properties of the query graph and eliminating infeasible join combinations early in the search process, HYPERION-Q reduces the planning time to only 35 milliseconds.

This represents a speedup of approximately 34× compared to the traditional DPccp

Citation: Vangala. R. Reddy, Praveen Kumar Kumbum (2024) HYPERION-Q: A Graph-Guided Self-Validating and Hardware-Aware Framework for High-Performance Query Processing. Journal of Data Science and Information Technology, 1(2), 1–7 doi: https://dx.doi.org/10.55124/jdit.v1i2.282

approach. The primary reason for this improvement is that the query graph representation enables the optimizer to avoid exploring join subsets that lack valid join predicates.

Consequently, the effective search space is drastically reduced, particularly for sparse query graphs commonly found in analytical workloads.

Bug Detection Capability

The second experiment evaluates the effectiveness of the transformation-invariant query testing mechanism integrated into HYPERION-Q. The goal is to determine how well the system can detect logical inconsistencies in query execution results.

Table 3 Bug Detection Capability

Method	Bugs Detected	Detection Rate
Random Testing	9	40%
AEI	18	70%
HYPERION-Q	26	92%

Traditional random query testing methods detected only nine logic bugs during the testing campaign. This relatively low detection rate is expected because random testing rarely generates semantically equivalent queries that can expose inconsistencies.

The Affine Equivalent Inputs (AEI) approach significantly improves bug detection capability by generating geometrically equivalent queries that should produce identical results. This method detected 18 logic bugs, corresponding to a detection rate of approximately 70%.

The HYPERION-Q framework further enhances this approach by incorporating multiple transformation strategies, including affine transformations, predicate rewrites, join reordering, and projection equivalence. As a result, the system detected 26 unique logic bugs, achieving a detection rate of 92%.

Another advantage of the HYPERION-Q validation mechanism is that it operates dynamically during query execution. This allows the system to identify errors in real time rather than relying solely on offline testing procedures. Consequently, the framework provides stronger correctness guarantees for mission-critical applications such as geographic information systems and financial data analytics.

Query Execution Throughput

The third experiment measures the performance of the hardware-aware execution scheduler. The objective is to evaluate how effectively HYPERION-Q utilizes heterogeneous hardware resources to accelerate query processing.

Table 4 Query Processing Throughput

System	Throughput (GB/s)	Latency (ms)
CPU Engine	18	210
GPU Engine	60	110
HYPERION-Q	100	75

The traditional CPU-based query execution engine achieved a throughput of 18 GB/s with an average latency of 210 milliseconds. While CPU execution provides flexibility and low overhead for control-intensive operations, it cannot fully exploit parallelism for large-scale analytical queries.

The GPU-based execution engine significantly improves performance by leveraging massive parallelism, achieving a throughput of approximately 60 GB/s. However, GPU execution alone does not always yield optimal results because certain operations, such as index lookups and control-heavy tasks, perform better on CPUs.

The HYPERION-Q framework integrates both CPU and GPU execution while also utilizing NVMe storage for efficient data spilling. By dynamically assigning operators to the most appropriate hardware resource, the system achieves a throughput of 100 GB/s with a latency of only 75 milliseconds.

This improvement demonstrates the effectiveness of the hardware-aware scheduling strategy. Compute-intensive operations such as joins and aggregations are executed on GPUs, while control-intensive tasks are handled by CPUs. Additionally, NVMebased spilling ensures that large intermediate results do not degrade performance when memory limits are reached.

Discussion

The experimental results demonstrate that HYPERION-Q provides substantial improvements across multiple aspects of database system performance. The graphguided optimizer dramatically reduces query planning time, enabling efficient optimization even for complex analytical queries. The transformation-invariant validation mechanism significantly enhances system reliability by detecting logic bugs that would otherwise remain unnoticed. Finally, the hardware-aware execution scheduler effectively exploits heterogeneous computing resources to maximize query processing throughput.

Overall, the results confirm that integrating optimization, correctness validation, and hardware-aware execution within a unified framework can significantly improve both the performance and reliability of modern data processing systems.

Conclusion and Future Work

This paper introduced HYPERION-Q, a novel hybrid framework for efficient and reliable query processing. By combining graph-based optimization, transformationinvariant testing, and hardware-aware execution planning, the framework significantly improves both performance and reliability.

Future work will explore integrating machine learning models for adaptive cost estimation and extending the framework to distributed cloud data platforms.

References

1 . Zheng, L., Xiao, Q., Cai, X.: A universal sketch for estimating heavy hitters and per-element frequency moments in data streams with bounded deletions. Proceedings of the ACM on Management of Data 2(6), 1–28 (2024)

2. Xu, X., Liu, H., Lv, X., Wang, Y., Li, D.: An efficient and exact algorithm for locally h-clique densest subgraph discovery. Proceedings of the ACM on Management of Data 2(6), 1–26 (2024)

3. Yao, Y., Chen, L., Fang, Z., Gao, Y., Jensen, C.S., Li, T.: Camel:

Citation: Vangala. R. Reddy, Praveen Kumar Kumbum (2024) HYPERION-Q: A Graph-Guided Self-Validating and Hardware-Aware Framework for High-Performance Query Processing. Journal of Data Science and Information Technology, 1(2), 1–7 doi: https://dx.doi.org/10.55124/jdit.v1i2.282

Efficient compression of floating-point time series. Proceedings of the ACM on Management of Data 2(6), 1–26 (2024)

4. He, Y., Wang, K., Zhang, W., Lin, X., Zhang, Y.: Common neighborhood estimation over bipartite graphs under local differential privacy. Proceedings of the ACM on Management of Data 2(6), 1–26 (2024)

5. Liu, Y., Zhao, S., Liu, Y., Zhao, D., Chen, H., Li, C.: Collecting triangle counts with edge relationship local differential privacy. In: 2022 IEEE 38th International Conference on Data Engineering (ICDE), pp. 2008–2020 (2022). IEEE

6. Liu, X., Oh, S.: Minimax optimal estimation of approximate differential privacy on neighboring databases. Advances in neural information processing systems 3 (2019)

7. Yu, L., Ye, Q., Du, R.: Privrm: A framework for range mean estimation under local differential privacy. Proceedings of the ACM on Management of Data 3(3), 1–26 (2025)

8. Liu, C., Chakraborty, S., Mittal, P.: Dependence makes you vulnberable: Differential privacy under dependent tuples. In: NDSS, vol. 16, pp. 21–24 (2016)13

9. Liu, Y., Wang, T., Liu, Y., Chen, H., Li, C.: Edge-protected triangle count estimation under relationship local differential privacy. IEEE Transactions on Knowledge and Data Engineering 36(10), 5138–5152 (2024)

10. Hu, Y., Ding, H., Dong, W.: N2e: A general framework to reduce node-differential privacy to edge-differential privacy for graph analytics. Proceedings of the ACM on Management of Data 3(6), 1–26 (2025)

11. Mohapatra, S., Gilad, A., He, X., Kimelfeld, B.: Computing inconsistency measures under differential privacy. Proceedings of the ACM on Management of Data 3(3), 1–27 (2025)

12. Li, C., Ni, W., Ding, M., Qu, Y., Chen, J., Smith, D., Zhang, W., Rakotoarivelo, T.: Decentralized privacy preservation for critical connections in graphs. IEEE Transactions on Knowledge and Data Engineering 36(11), 5911–5925 (2024)

13. Fang, J., Dong, W., Yi, K.: Shifted inverse: A general mechanism for monotonic functions under user differential privacy. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pp. 1009–1022

14. Ye, Q., Hu, H., Au, M.H., Meng, X., Xiao, X.: Towards locally differentially private generic graph metric estimation. In: 2020 IEEE 36th International Conference on Data Engineering (ICDE), pp. 1922–1925 (2020). IEEE

15. Chen, R., Fung, B.C., Yu, P.S., Desai, B.C.: Correlated network data publication via differential privacy. The VLDB Journal 23(4), 653–676 (2014)

16. Li, C.: Optimizing Linear Queries Under Differential Privacy. University of Massachusetts Amherst, ??? (2013)

17. Ye, Q., Hu, H., Au, M.H., Meng, X., Xiao, X.: Lf-gdpr: A framework for estimating graph metrics with local differential privacy. IEEE Transactions on Knowledge and Data Engineering 34(10), 4905–4920 (2020)

18. Sun, H., Xiao, X., Khalil, I., Yang, Y., Qin, Z., Wang, H., Yu, T.: Analyzing subgraph statistics from extended local views with decentralized differential privacy. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 703–717 (2019)

19. Stausholm, N.M.: Improved differentially private euclidean distance approximation. In: Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, pp. 42–56 (2021)

20. Gonzalez, J.E., Low, Y., Gu, H., Bickson, D., Guestrin, C.: {PowerGraph}: Distributed {Graph-Parallel} computation on natural graphs. In: 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12), pp. 14 17–30 (2012)

21. Han, W.-S., Lee, S., Park, K., Lee, J.-H., Kim, M.-S., Kim, J., Yu, H.: Turbograph: a fast parallel graph engine handling billion-scale graphs in a single pc. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 77–85 (2013)

22. Stoian, M., Kipf, A.: Dpconv: Super-polynomially faster join ordering. Proceedings of the ACM on Management of Data 2(6), 1–26 (2024)

23. Deng, W., Mang, Q., Zhang, C., Rigger, M.: Finding logic bugs in spatial database engines via affine equivalent inputs. Proceedings of the ACM on Management of Data 2(6), 1–26 (2024)

24. Yan, M., Wang, Y., Wang, Y., Miao, X., Li, J.: Gidcl: A graph-enhanced interpretable data cleaning framework with large language models. Proceedings of the ACM on Management of Data 2(6), 1–29 (2024)

25. Boeschen, N., Ziegler, T., Binnig, C.: Golap: A gpu-in-data-path architecture for high-speed olap. Proceedings of the ACM on Management of Data 2(6), 1–26 (2024)

26. Kuschewski, M., Giceva, J., Neumann, T., Leis, V.: High-performance query processing with nvme arrays: Spilling without killing performance. Proceedings of the ACM on Management of Data 2(6), 1–27 (2024)

Citation: Vangala. R. Reddy, Praveen Kumar Kumbum (2024) HYPERION-Q: A Graph-Guided Self-Validating and Hardware-Aware Framework for High-Performance Query Processing. Journal of Data Science and Information Technology, 1(2), 1–7 doi: https://dx.doi.org/10.55124/jdit.v1i2.282