IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

An Overview of Query Optimization in Distributed Relational Databases

Abhayanand[1]
Research Scholar, PG Department of CS, Patliputra University, Patna, Bihar, India

Dr. M. M. Rahman[2]

Associate Professor, PG Department of Mathematics, A. N. College, Patna, Bihar, India

Abstract:

In the domain of distributed relational databases, efficient query optimization stands as a cornerstone for achieving peak performance. As data spans across multiple nodes, the intricacy of query handling escalates, emphasizing the pivotal role of optimization methods in curtailing execution time and resource consumption. This paper presents a thorough examination of query optimization strategies within distributed relational databases, encompassing both centralized and distributed approaches. We explore a spectrum of techniques, such as cost-based optimization, parallel query processing, and adaptive query processing, elucidating their advantages, constraints, and suitability across diverse distributed database frameworks. Moreover, we address emerging trends and hurdles in query optimization within distributed systems, offering insights into potential paths for future research and advancement.

Keywords: Query Optimization, Relational Databases, Partitioning, Challenges

1. Introduction: In the age of big data and distributed computing, handling immense amounts of data across distributed environments presents considerable hurdles. Distributed relational databases have become fundamental to contemporary data management, enabling organizations to store and analyze data across numerous nodes with scalability, fault tolerance, and high availability. Yet, as distributed databases expand in size and intricacy, optimizing query performance becomes ever more crucial to meet the requirements of modern applications and users. Instead of showcasing Query Optimizer works in DBMS as illustrated in Figure 1, emphasis is now placed on enhancing query performance.

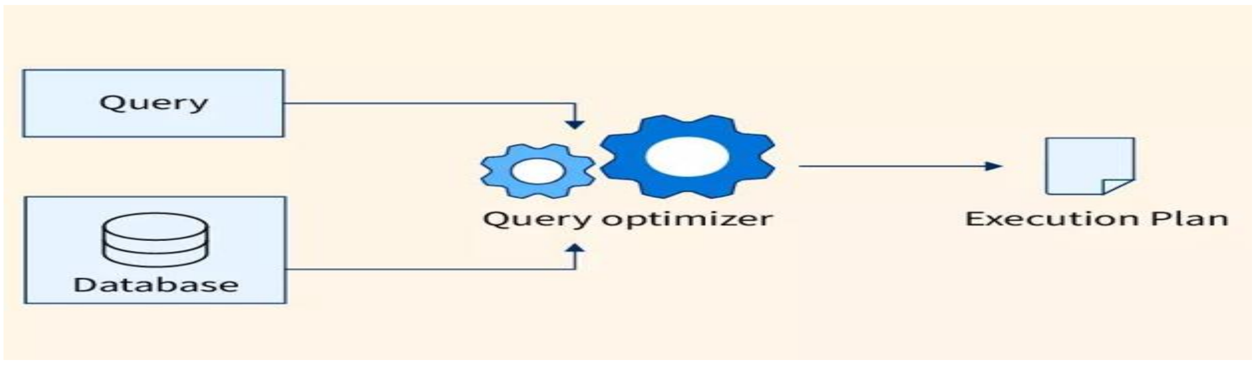

Figure 1: Query Optimizer works in DBMS

The primary goal of query optimization in distributed relational databases is to maximize throughput and scalability while decreasing response times and resource consumption. In contrast to traditional centralized databases, which store data on a single node, distributed databases share data across several nodes, frequently located in several geographic areas. As a result, running queries in distributed systems creates special optimization challenges since it requires navigating network links, coordinating data access across nodes, and balancing processing resources. The field of distributed relational database query optimization includes a wide range of methods and approaches designed to improve query execution performance. These techniques range from more complex distributed optimization algorithms designed for distributed systems to more conventional centralized optimization techniques. Furthermore, the emergence of parallel processing, adaptive optimization, and hybrid optimization techniques adds further complexity, offering new avenues for enhancing query performance in distributed systems. This paper offers a comprehensive overview of query optimization in distributed relational databases, delving into the foundational principles, techniques, challenges, and emerging trends in the field.

2. Fundamentals of Distributed Relational Databases: Dispersed relational databases are essential to modern data management because they let businesses effectively store, read, and handle enormous volumes of data in a variety of dispersed computer contexts. Figure 2 illustrates how distributed relational databases disperse data among several nodes connected by a network, in contrast to conventional centralized databases, which store data on a single server. The distributed architecture has several benefits, such as fault tolerance, scalability, and enhanced performance via parallel processing.

Figure 2: Distributed Relational Databases of Supercomputer (Source: Google)

2.1 Overview of Distributed Database Architecture: A network of linked nodes, each hosting a component of the database, makes up distributed relational databases. These nodes enable enterprises to store and analyze data closer to the point of generation or demand by being geographically dispersed across many locations. Because the database design is distributed, it allows for horizontal scalability, which allows for the addition of nodes to the cluster in response to increasing workload needs and data volumes.

A distributed relational database's architecture usually consists of metadata servers, data nodes, query processing nodes, and communication protocols. While query processing nodes organize query execution and data retrieval throughout the distributed environment, data nodes actually store and maintain the data.

2.2 Challenges in Distributed Query Processing: Although distributed relational databases have many advantages, query processing and optimization present particular difficulties. Data distribution, which involves dividing or replicating data among several nodes for enhanced performance and fault tolerance, is one of the main obstacles. since of this spread, designing and executing searches becomes more complicated since queries may need to access data that is located on various nodes, necessitating network coordination and data sharing.

One such difficulty is in load balancing, which involves distributing query processing work uniformly among nodes to prevent bottlenecks and optimize resource use. Different data access patterns and query workloads in a distributed context might result in unequal resource use, which can impact system performance as a whole.

Distributed databases also have to deal with problems of fault tolerance, transaction management, and data consistency. Maintaining data integrity and system resilience in distributed systems requires coordinating distributed transactions, ensuring consistency across distributed data copies, and gracefully resolving errors.

2.3 Importance of Query Optimization: Achieving effective and scalable database operations requires enhancing query performance due to the inherent complexity of distributed query processing. Query optimization chooses the best execution plans based on variables including data distribution, access patterns, and system resources in order to reduce the amount of time and resources needed to run queries. Query optimization in distributed relational databases combines centralized and distributed optimization methods. The goal of centralized optimization is to create the best query execution plans possible while taking into account the availability of resources on each node and the local data access patterns. In contrast, distributed optimization takes into consideration the global query execution plan over several nodes while taking inter-node communication costs, network latency, and data distribution into account. A thorough grasp of the underlying database architecture, query processing methods, and optimization techniques is necessary for efficient query optimization in distributed relational databases. Organizations may enhance the speed and scalability of their distributed database systems and facilitate effective data management and analysis in intricate distributed computing settings by utilizing sophisticated optimization techniques and technologies. The core concepts of distributed relational databases, such as their design, difficulties, and the significance of query optimization, are covered in greater detail in this enlarged section.

3. Centralized Query Optimization: A key component of query processing in distributed relational databases is centralized query optimization, in which query execution plans are created and optimized within the specific context of each database node. Centralized query optimization generates optimal plans based on local data access patterns, system resources, and query limitations; in contrast to distributed query optimization, which takes into account the global query execution plan across numerous nodes.

3.1 Basic Principles and Techniques: A collection of basic ideas and methods for creating effective query execution plans are the foundation of centralized query optimization. These methods consist of plan creation, cost estimate, transformation, and query parsing. The database optimizer parses a query after it is received to determine its constituent parts, including tables, predicates, and join criteria. The query is then converted into an internal representation, which makes it easier to analyze and optimize later on, such a query tree or a relational algebra expression. Cost estimate, which involves calculating the execution costs of various query execution plans based on variables including data distribution, access techniques, and join strategies, is essential to centralized query optimization.

3.2 Query Parsing and Transformation: The optimizer examines the query structure and rewrites it into forms that are semantically comparable and amenable to optimization throughout the query parsing and transformation process. Depending on query semantics and optimization criteria, this might entail rearranging predicates, implementing query transformations, or choosing suitable access methods and join techniques. To increase performance, the optimizer could, for instance, break down a complicated join query into a series of easier join operations or take advantage of materialized views and indexes for effective data access.

3.3 Cost Estimation and Plan Generation: In centralized query optimization, cost estimate entails determining the resource needs and execution times of various query execution strategies. The optimizer estimates the cost of obtaining and processing data using different access pathways and join algorithms by using statistical information such as table cardinalities, index selectivities, and join selectivities. The optimizer creates candidate execution plans based on these cost estimates, then chooses the plan with the lowest projected cost to execute. When data distribution among nodes is reasonably homogeneous or when queries primarily access data that resides on a single node, centralized query optimization approaches are well-suited. Centralized optimization techniques might not, however, completely take advantage of the parallelism and distributed structure of the system in distributed relational databases, where data may be dispersed over several nodes with differing access costs and network latencies.

Although it has drawbacks, centralized query optimization is nevertheless a vital part of query processing in distributed relational databases as it offers fundamental methods and guidelines for creating effective query execution plans. Organizations can get optimal query performance in intricate distributed database settings by utilizing both distributed optimization methodologies and centralized optimization approaches.

4. Distributed Query Optimization: In distributed relational databases, distributed query optimization is a crucial part of query processing that aims to provide effective query execution plans that span numerous nodes in the distributed context. Distributed query optimization takes into consideration the global query execution plan over numerous nodes, accounting for data distribution, network latency, and inter-node communication costs. This is in contrast to centralized query optimization, which concentrates on optimizing queries within the context of individual nodes.

4.1 Challenges Specific to Distributed Environments: Because the database system is dispersed, there are special difficulties with distributed query optimization. Data distribution, which involves dividing or replicating data among several nodes for fault tolerance and scalability, is one of the main obstacles. In these kinds of settings, query optimization necessitates decreasing data transmission and communication overhead while coordinating data access and processing among remote nodes.

Heterogeneity is a further problem in distributed systems, because nodes may differ in terms of computing power, network bandwidth, and hardware setups. Query execution plans must be dynamically adjusted to optimize queries in heterogeneous settings. This allows for the best use of each node's resources, balanced task distribution, and maximum system performance.

Furthermore, data skew problems, in which data is distributed unevenly among nodes and causes resource contention and performance bottlenecks, must be addressed in distributed query optimization. Data partitioning algorithms, query parallelization, and dynamic load balancing are some of the approaches needed to address data skew and guarantee effective query processing and resource usage across dispersed nodes.

4.2 Cost-Based Optimization in Distributed Settings: Distributed query optimization is based on cost-based optimization, in which the optimizer assesses the cost of various query execution plans by taking into account inter-node communication costs, data distribution, and access mechanisms. Distributed optimization needs global optimization statistics and cost estimates to provide the best query plans over several nodes, in contrast to centralized optimization, which depends on local statistics and cost models.

Cost-based optimization takes into account not just the computing expenses related to processing individual query operators in distributed systems, but also the costs of data transport and communication between nodes. To precisely estimate the cost of processing query fragments on various nodes and coordinating data flow across them, this entails assessing network latencies, data transfer rates, and inter-node communication overhead.

4.3 Global Query Optimization vs. Local Query Optimization: Both local and global optimization strategies are included in distributed query optimization, and they each focus on a distinct facet of query processing in distributed systems. The goal of global optimization is to create the best possible query execution plan over a number of nodes while taking inter-node communication costs, data placement tactics, and data distribution into account. By using parallelism and data locality across distributed nodes, global optimization seeks to reduce overall query execution time and resource use.

On the other hand, local optimization strategies concentrate on query execution plan optimization within the specific environment of particular nodes, taking into account processing capabilities, index availability, and local data access patterns. The goal of local optimization is to maximize local processing parallelism, ensure effective resource usage, and reduce the cost of running query fragments on each node.

Distributed query optimizers may provide thorough query execution plans that take advantage of the distributed structure of the database system while enhancing efficiency and scalability by combining global and local optimization approaches. To optimize queries efficiently across distributed systems, one must possess a thorough comprehension of the underlying database architecture, query semantics, and optimization techniques. Additionally, one must be flexible enough to adjust to shifting workloads and system conditions.

This extended part offers a comprehensive analysis of distributed query optimization, emphasizing the difficulties, methods, and factors to be taken into account when creating effective query execution plans across distributed relational databases.

5. Parallel Query Processing: By using the processing power of several nodes at once, parallel query processing is essential to increasing the scalability and performance of distributed relational databases. This section explores in further detail certain facets of parallel query processing, including as resource management, partitioning schemes, and execution plans.

5.1 Parallel Execution Strategies: Parallel query processing involves dividing a query into subtasks that can be executed simultaneously across multiple nodes. There are several strategies for parallelizing queries:

- Pipeline Parallelism: In pipeline parallelism, several nodes carry out distinct phases of query processing, such as scanning, filtering, and aggregation, simultaneously. Pipelined execution is made possible by each node processing a piece of the data and sending the intermediate results to the next step.
- Partition Parallelism: This technique entails partitioning the data and allocating each partition to a distinct processing node. This technique works especially well for queries (like range queries or join operations) that can be broken down into discrete processes.
- Operator Parallelism: Within a query execution plan, operator parallelism focuses on parallelizing specific operators. One way to parallelize a join process is to divide the input relations into multiple partitions and execute the join operations concurrently on each partition.
- Task Parallelism: This technique divides a query into smaller jobs that may be carried out separately by many nodes. To effectively utilize parallel processing capabilities, each job is given to a node according to its capacity and availability.

5.2 Partitioning Techniques for Parallelism: In parallel query processing, load balancing and lowering communication overhead depend on effective data partitioning. Typical methods for partitioning include:

- Hash Partitioning: This technique divides the data across several nodes by hashing it according to a selected property. This makes sure that tuples on the same node are saved with the same hash value, which makes join and aggregation operations easier to complete in parallel.
- Range Partitioning: Based on a given property, such a date or numerical value, range partitioning separates the data into discrete ranges. Range-based queries may be processed efficiently in parallel since each range is assigned to a separate node.
- Round-robin partitioning: It is a technique that divides data tuples in an even and circular manner across nodes.

- Composite Partitioning: To get the best possible data distribution, composite partitioning uses many partitioning approaches. To balance query execution efficiency, a hybrid solution may, for big tables, employ hash partitioning and, for smaller tables, range partitioning.

5.3 Load Balancing and Resource Allocation: Optimizing the speed of simultaneous query processing requires effective load balancing and resource allocation. Load balancing prevents idle nodes and resource constraints by distributing computing jobs equally across nodes. Workload-aware scheduling and adaptive task allocation are two examples of dynamic load balancing strategies that can assist maximize resource efficiency and reduce query execution time.

Allocation of resources is the process of assigning memory and CPU cores to query processing jobs in accordance with their priorities and needs. The effective use of resources is made possible by strategies like resource sharing and dynamic resource provisioning, which also guarantee performance isolation and fairness across concurrent queries.

To summarize, by utilizing parallelism and spreading query execution over several nodes, parallel query processing provides notable performance advantages in distributed relational databases. Achieving optimal performance and scalability in distributed database systems requires the use of effective resource management strategies, data partitioning techniques, and parallel execution algorithms.

6. Adaptive Query Processing: With the help of adaptive query processing, database systems may dynamically optimize their queries by modifying their execution plans in response to runtime feedback and shifting environmental factors. Adaptive query processing approaches, in contrast to traditional static optimization techniques, improve query performance and resource consumption by adjusting to variations in data distribution, system load, and query parameters. The significance of many adaptive query processing techniques in the context of distributed relational databases is examined in this section.

- Dynamic Plan Modification: Adaptive query processing's core tenet is the flexibility to dynamically modify query execution strategies in real time. This entails keeping an eye on the query execution process and making judgments in real time to adjust the execution strategy in light of performance indicators that are noticed.
- Query Feedback Mechanisms: The collection of runtime data and performance measurements in adaptive query processing is dependent on feedback mechanisms. These approaches might be resource use monitoring, query tracing, and runtime profiling. The database system can spot abnormalities, pinpoint performance bottlenecks, and make wise judgments to maximize query execution by examining feedback data. For example, the system can dynamically construct or eliminate indexes to increase performance if it is discovered that a certain index is useless for a particular query pattern.
- Algorithms for Adaptive Joins: Query processing efficiency is sometimes hampered by join procedures, particularly in remote contexts with big datasets. Adaptive join algorithms, which dynamically choose the best join method based on runtime conditions, are included in adaptive query processing approaches.
- Runtime Query Optimization: Adaptive query processing uses runtime optimization strategies to enhance performance in addition to modifying query execution plans. This might involve adaptive parallelism management, dynamic index selection, and dynamic data partitioning. The system may automatically prune partitions depending on query predicates, for instance, if a query uses range predicates on a partitioned table. This will lower the volume of data processed and improve query speed.

- Caching and Reusing Query Plans: To improve performance and scalability, adaptive query processing can also make advantage of query plan caching and reuse. The system can accelerate query processing and eliminate duplicate optimization costs by storing and reusing query plans from prior executions. The system must, however, adaptively invalidate and update stored query plans in response to guarantee efficacy.

Because adaptive query processing allows for the dynamic modification of execution plans in response to runtime feedback and shifting conditions, it is essential for enhancing query performance in distributed relational databases. Distributed database systems can work more efficiently and effectively in unpredictable and dynamic environments by using adaptive techniques like query plan caching, adaptive join algorithms, dynamic plan adjustment, runtime query optimization, and query feedback mechanisms.

7. Hybrid Query Optimization: A hybrid strategy that combines the advantages of distributed and centralized optimization approaches has gained interest in the quest for optimal query performance in distributed relational databases. The goal of hybrid query optimization is to minimize the drawbacks of both distributed and centralized approaches while utilizing their respective benefits. In-depth discussion of the elements, methods of implementation, and possible advantages of hybrid query optimization is provided in this section.

7.1 Combining Distributed and Centralized Optimization Methods: In a distributed database management system (DBMS), hybrid query optimization entails the seamless integration of centralized and distributed optimization approaches. Fundamentally, this strategy combines the scalability and adaptability provided by distributed optimization techniques with the effectiveness of centralized query optimization in producing high-quality query plans.

Query parsing, query transformation, and cost estimate are examples of centralized optimization techniques that are excellent at creating the best execution plans for specific requests. They could, however, ignore the inter-node communication costs and global system properties that come with distributed settings. dispersed optimization approaches, on the other hand, take into account the dispersed nature of data and the possibilities of parallel processing; nevertheless, they may have more complexity and communication cost. Hybrid query optimization combines distributed and centralized optimization strategies to try to balance scalability and query plan quality. Usually, this integration entails:

- Query Plan Generation: Based on local data and query predicates, candidate query plans are generated via centralized optimization.
- Cost-Based Adjustment: By taking into account network latency, data dispersion, and global system statistics, the resulting query plans are improved by using distributed optimization concepts.
- Runtime Adaptation: Using feedback methods to constantly enhance performance, query execution techniques are dynamically adjusted based on workload fluctuations and system circumstances.

7.2 Models for Hybrid Query Execution: A variety of execution models that combine distributed and centralized processing paradigms are included in hybrid query optimization. Typical hybrid query execution models include the following:

- Partitioned Approach: Dividing the query execution procedure into phases that are spread and centrally located. To create an initial plan, the query is first optimized centrally. The plan is then improved by taking communication costs and data dispersion into consideration through the use of distributed optimization techniques.
- Parallelization with Centralized Planning: Making use of centralized planning for query optimization while taking advantage of parallel query processing capabilities. This strategy seeks to take use of

distributed settings' parallelism while gaining from centralized techniques' greater optimization powers.

- Dynamic Plan Selection: Based on query properties, data distribution, and system workload, this technique dynamically chooses between centralized and distributed query execution techniques. With this adaptive technique, the system may optimize performance based on the current circumstances by selecting the best optimization strategy for each query at runtime.

7.3 Trade-Offs and Benefits:

There are several advantages and trade-offs with hybrid query optimization.

- Performance Optimization: Compared to depending only on one strategy, hybrid approaches can achieve better query performance by mixing centralized and distributed optimization techniques. Complex queries that necessitate striking a balance between local and global optimization factors would particularly benefit from this optimization.
- Flexibility and Scalability: Hybrid optimization preserves flexibility to adjust to changing workload patterns and system requirements while enabling scalable query processing in distributed situations.
- Complexity Management: Although hybrid optimization improves efficiency, it also adds more complexity to the planning and execution of queries. Hybrid optimization techniques must be carefully designed and put into practice in order to manage this complexity.
- Resource consumption: The goal of hybrid optimization is to maximize computing efficiency while reducing network overhead by optimizing resource consumption across distributed nodes. In conclusion, a viable strategy for attaining effective query processing in distributed relational databases is hybrid query optimization. Hybrid methods have the potential to improve query performance, scalability, and flexibility in distributed database settings by combining centralized and distributed optimization techniques.

8. New Trends and Difficulties

8.1 Distributed Query Optimization and Big Data: Big data's widespread use has made distributed databases that can manage enormous amounts of data across dispersed clusters necessary. Because of the volume and complexity of the data, query optimization in the context of big data presents new difficulties that call for cutting-edge optimization strategies. To improve searches across massively dispersed datasets, methods like data partitioning, distributed caching, and parallel processing are being used more and more. There are chances to leverage big data frameworks' optimization capabilities within distributed relational databases by integration with them, such as Apache Hadoop and Apache Spark.

8.2 Machine Learning Methods for Optimizing Query Structure: Neural networks, reinforcement learning, evolutionary algorithms, and other machine learning approaches show promise for improving query optimization in distributed databases.

Machine learning models are able to adaptively improve query plans to fit particular workload patterns and system settings by learning from past query execution data and system performance parameters.

Effective learning model design, scalability to large-scale distributed systems, and the interpretability of learnt query optimization algorithms are among the challenges.

8.3 Privacy and Security Considerations: In light of the growing significance of data privacy and security, query optimization strategies need to take into account issues with sensitive data access and regulatory compliance (e.g., GDPR, HIPAA). To provide privacy-preserving query processing in distributed contexts, strategies including secure multi-party computing, differential privacy, and query rewriting are being investigated. Finding a balance between data security and query performance is still a major difficulty, especially when stringent privacy regulations need to be followed.

8.4 Scalability and Fault Tolerance:

- Query optimization techniques must be designed to function well in large-scale distributed systems with hundreds or thousands of nodes while also providing resilience.
- As distributed databases scale to accommodate growing data volumes and user demands, scalability and fault tolerance become paramount.

In the event of a breakdown, strategies like data shading, replication, and fault-tolerant query processing are crucial for preserving system speed and availability.

8.5 Integration and Interoperability Across Domains:

An important difficulty in heterogeneous dispersed settings is ensuring smooth integration and interoperability, as data may be spread across several databases and platforms.

- The complexity of federated querying, in which queries span several data sources with different schemas and processing capacities, requires query optimization approaches to take these differences into account.
- Standardization initiatives, including the SQL/MED (SQL Management of External Data) standard and SQL:1999's support for distributed querying, are meant to make distributed databases more interoperable, but they still need to be improved upon to solve real-world issues.

8.6 Energy Efficiency and Green Computing:

- The optimization of query processing to reduce energy usage and advance green computing techniques is becoming more popular as environmental concerns develop.
- The carbon footprint of distributed database systems can be decreased by utilizing strategies like workload consolidation, dynamic resource provisioning, and energy-aware scheduling; however, doing so involves trade-offs that must be carefully considered and creatively resolved in order to optimize for energy efficiency while maintaining query performance.

In order to handle the complexity of contemporary distributed computing settings, research and innovation must continue. These new trends and challenges illustrate the changing landscape of query optimization in distributed relational databases.

Conclusion:

To sum up, this study has examined the complexities of query optimization in distributed relational databases, illuminating the underlying ideas, methods, and difficulties in this vital field of database administration. Our thorough examination of centralized and distributed query optimization techniques—such as adaptive and parallel query processing—has shown how crucial it is to optimize query execution in order to improve performance and resource management in distributed settings.

The dynamic environment of distributed systems, marked by the spread of large data and the introduction of new computing paradigms, offers query optimization prospects as well as difficulties. New methods, such machine learning-based optimization and adaptive query processing, are gaining popularity and show promise in addressing the scalability issue with distributed databases, even as older techniques continue to play a significant role.

There are a number of directions that query optimization research and development might go in the future. Exploring hybrid optimization strategies, which combine the advantages of distributed and centralized methods, might lead to the best results in heterogeneous distributed systems. Furthermore, enhancing fault tolerance capabilities and integrating privacy-preserving measures are essential for guaranteeing the security and dependability of distributed database systems.

In order to spur innovation and advance the state-of-the-art in query optimization for distributed relational databases, it is essential that researchers, practitioners, and industry stakeholders work closely together. Through the promotion of multidisciplinary cooperation and the adoption of novel technologies, we may fully use distributed computing and open up new avenues for data-intensive applications and analytics.

As it wraps up, this work is a call to action for more research, development, and cooperation in the area of distributed database query optimization. We can create database systems that are more durable, scalable, and efficient by tackling the problems and taking advantage of the possibilities that come with distributed computing. This will enable businesses to get the most out of their data assets in the digital era.

References:

1. Agrawal, S., & Abbadi, A. E. (Eds.). (2013). Query Processing in Distributed Database Systems (Vol. 10). Springer Science & Business Media.

2. Chaudhuri, S., & Narasayya, V. (1998). An efficient cost-driven index selection tool for Microsoft SQL Server. In Proceedings of the 24th International Conference on Very Large Data Bases (VLDB) (pp. 146-155).

3. DeWitt, D. J., & Gray, J. (1992). Parallel database systems: the future of high performance database systems. Communications of the ACM, 35(6), 85-98.

4. Graefe, G. (1993). Query evaluation techniques for large databases. ACM Computing Surveys (CSUR), 25(2), 73-170.

5. Hadoop Apache. (n.d.). Apache Hadoop. Retrieved from https://hadoop.apache.org/

6. He, Y., Lee, R., Huai, Y., Shao, Z., Jain, N., & Zhang, X. (2011). RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (pp. 1199-1210).

7. Lohman, G. M. (1988). Grammar-like functional rules for representing query optimization alternatives. In Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data (pp. 18-27).

8. Olston, C., Reed, B., Srivastava, U., Kumar, R., & Tomkins, A. (2008). Pig Latin: A not-so-foreign language for data processing. In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (pp. 1099-1110).

9. Stonebraker, M., & Çetintemel, U. (2005). "One size fits all": An idea whose time has come and gone. In Proceedings of the 21st International Conference on Data Engineering (ICDE) (pp. 2-11).

10. Thapliyal, H., & Rathi, S. (2015). A survey on query optimization techniques in distributed databases. International Journal of Computer Applications, 124(5), 1-5.

11. Valduriez, P., & Elmagarmid, A. K. (1995). Join processing in relational databases. ACM Transactions on Database Systems (TODS), 20(2), 197-260.

12. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., ... & Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI) (pp. 2-2).