Journal of Global Research in Multidisciplinary Studies (JGRMS)
Volume 2 | Issue 10 | October 2026 | Pages 22–29
DOI: https://doi.org/xxxxxxxxxxxxx
e-ISSN: 3049-2963
Review Article OPEN ACCESS Available online: https://saanvipublications.com/journals/index.php/jgrms

A Review of Data Caching and Query Acceleration Techniques in Modern Data Warehouse Systems

Mr. Kapil Ahir
Assistant Professor
Department of Computer Sciences and Applications
Mandsaur University, Mandsaur
kapil.ahir@meu.edu.in

Corresponding Author: * Kapil Ahir (kapil.ahir@meu.edu.in)

Received: 10 October, 2026 | **Revised:** 15 October, 2026 | **Accepted:** 28 October, 2026

***Abstract*—Data warehouse systems face increasing challenges in processing large-scale analytical queries efficiently due to growing data volumes, complex workloads, and resource constraints. Various data caching and query acceleration methods are presented that may be useful to decrease query latency, reduce redundant computation, and improve resource utilization. This paper briefly summarizes the major techniques like query-result caching, materialized views, in-memory caching, indexing, data partitioning, parallel processing, and hardware-assisted acceleration. It also explores workload-aware caching, adaptive query execution, reinforcement learning query optimization, and integrated cache-execution management in the cloud and distributed data warehouse settings. The analyzed papers demonstrate the advantages of GPU acceleration, distributed caching, cost-aware query processing, and adaptive cache replacement to enhance analytical throughput and query execution efficiency. The comparative analysis also points to some key challenges such as cache consistency, data freshness, dynamic workload adaptation, scalability, heterogeneous infrastructure, and infrastructure cost. The results highlight the importance of integrating cache management with query optimization and resource management for effective analytical processing. Based on this review, the following research gaps are identified that can be addressed through building integrated, workload-aware systems that satisfy the requirements of query performance, cache efficiency, data freshness, and operational cost for modern data warehouse systems.**

***Keywords*—Data Warehousing, Data Caching, Query Acceleration, Workload-Aware Optimization, Adaptive Query Processing, Distributed Caching, Resource Management.**

I. INTRODUCTION

A data warehouse is a collection of data that is stored in a single location for the purpose of analytical processing, reporting, and decision making, and that is integrated and consolidated from a variety of data sources. Data warehouses are optimized for complex queries [1], historical analysis, aggregation, and handling large volumes of data, whereas transactional databases are optimized for transactional data. In the fields of finance, healthcare, retail, manufacturing, telecommunications, and e-commerce, they are used extensively across various sectors for business intelligence and strategic decision-making [2].

Efficient query processing becomes a big challenge when data volume and complexity grow. Analytical queries are frequently large scans, joins, and aggregations that can take a long time to execute, introduce I/O overhead, and consume resources. The data caching solution is to store data objects that are often used, such as query results [3]intermediate data, or reusable data objects so that they can be accessed without additional calculation. Query-result caching, materialized views, and in-memory caching can thus minimize latency and resource consumption.

Query optimization [4] Indexing, partitioning, columnar storage, data skipping, parallel processing, and hardware-assisted execution are some query acceleration techniques that further enhance performance. With modern cloud and distributed warehouses, these techniques are frequently required to work together as workloads evolve. Thus, in this review, the focus is on information on data caching, query acceleration, workload-aware optimization, adaptive execution, as well as the combination of those topics in modern data warehouse environments [5]. In supply chain management, AI-driven big data analytics and cloud-integrated computing support the analysis of large volumes of operational data, highlighting the importance of efficient data processing and analytical performance in modern supply chain systems [6].

A. Structure of the Paper

The paper is organized as follows: Section II presents data caching techniques, Section III discusses query acceleration techniques, Section IV focuses on their integration, Section V reviews related literature, and Section VI concludes the paper with future research directions.

II. DATA CACHING TECHNIQUES FOR MODERN DATA WAREHOUSES

Data caching improves analytical performance by retaining frequently accessed query results and data objects for subsequent reuse. As illustrated in Figure 1 [7]. the system

Copyright © 2026 The Author(s).
This is an open access article distributed under the terms of the Creative Commons Attribution 4.0 International (CC BY 4.0) License (https://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, adaptation, and reproduction in any medium, provided the original author(s) and the source are properly credited.

DOI: https://doi.org/xxxxxxxxxxxxx

checks whether a valid result exists in the cache when a user submits a query. If a cache hit occurs, the stored result is returned directly; otherwise, the query is executed against the underlying database, and its result may be cached for future requests. Cache replacement policies, including Least Recently Used (LRU) and Least Frequently Used (LFU), determine which entries should be retained or evicted when cache capacity is limited. Size-aware and aging-based LFU strategies can additionally consider data size, access frequency, and entry age to improve cache utilization.

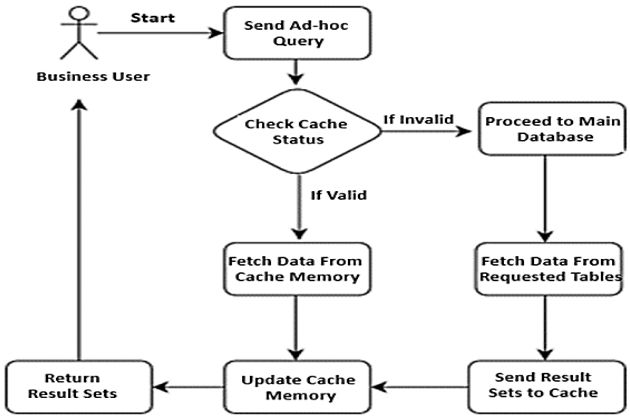


Fig. 1. Flow Chart of Cache Mechanism

A. *Fundamentals of Data Caching in Data Warehouses*

An important aspect of data caching in Online Analytical Processing (OLAP) systems is identifying when previously computed results can satisfy new analytical requests. OLAP queries frequently perform slicing, dicing, filtering, and aggregation over large fact tables, often producing relatively small result sets suitable for reuse [8]. However, traditional cache keys based on normalized SQL text or Abstract Syntax Tree (AST) representations may treat syntactically different but semantically equivalent queries as separate entries. Semantic caching addresses this limitation by considering measures, grouping levels, filters, time windows, and post-aggregation operations, thereby improving cache reuse across equivalent analytical requests. Using this method allows semantically similar SQL and natural language queries to use cached results for performance efficiency and to cut down on redundant computation, with the key components as follows:

- **Query Result Reuse:** Previously computed OLAP results are stored and reused for repeated analytical queries.
- **Reduced Backend Processing:** Cached results reduce repeated execution on the backend and lower computational requirements.
- **Cache Key Generation:** Traditional approaches identify cached results using normalized SQL text or AST-based representations.
- **Cache Fragmentation:** Different SQL formats and natural-language expressions can represent the same analytical request but generate different cache entries.
- **Semantic Caching:** Queries are represented using measures, grouping levels, filters, time windows, and post-aggregation operations to improve result reuse.

B. *Query Result Caching*

It requires effective mechanisms for matching incoming queries with stored results and maintaining their validity over time. When a compatible query matches a valid cache entry, the system can return the stored output without repeating the original computation [9].The effectiveness of this approach depends on query matching, cache consistency, invalidation, and replacement decisions. These mechanisms determine whether cached results remain reliable as underlying data and query requirements change. The key aspects of query-result caching are as follows:

- **Query-result caching:** Stores complete query results for fast retrieval.
- **Reuse of computed results:** Previously generated results can satisfy repeated or compatible queries without re-executing them.
- **Cache invalidation and consistency:** Cached results must be invalidated or refreshed when underlying data, schema, or relevant time partitions change to prevent stale results.
- **Result-cache management:** Includes cache updating, matching, replacement, and eviction policies such as LRU when storage capacity is reached.

C. *Materialized Views and View Caching*

In data warehouse systems, materialized views (MVs) accelerate analytical queries by persistently storing the results of frequently used or computationally expensive joins and aggregations. Unlike temporary query-result cache entries, materialized views provide reusable precomputed data structures that may serve multiple queries. Their selection depends on workload patterns, expected performance benefits, maintenance requirements, and storage costs. During query optimization, the system determines whether an available materialized view can satisfy all or part of a query and may rewrite the execution plan accordingly. Additional filtering or computation may be required to preserve the original query's semantics [10][11].

D. *In-Memory Data Caching*

High-speed memory is important in the context of data warehouse queries, because it speeds up queries by providing access to data that is used often. This is implemented using Processing-in-Memory (PIM) architecture, which incorporates computation units within DRAM devices [12]. This design enables memory-heavy Online Analytical Processing (OLAP) operations to run close to the data itself, thereby minimizing the expense of moving large amounts of data between memory and the CPU. In the meantime, CPU-based interleaved memory access is used for Online Transaction Processing (OLTP) operations. Applying these methods together helps to maximize memory bandwidth usage, minimize data transfer, and enable efficient operation of mixed transactional and analytical workloads.

- **In-memory processing:** PIM units execute memory-intensive OLAP operations directly inside DRAM, reducing CPU–memory data movement and exploiting the internal bandwidth of memory devices.
- **Buffer and memory caching:** PIM uses local SRAM buffers such as WRAM for data and IRAM for instructions. Data loaded into these buffers can be processed locally, reducing repeated DRAM accesses and improving execution efficiency.
- **Column-oriented caching:** Analytical workloads frequently access selected columns. PUSH tap organizes data so that columns can be efficiently accessed by PIM units, while its block-circulant

DOI: https://doi.org/xxxxxxxxxxxxx

placement distributes columns across PIM devices to improve parallel analytical processing.

- **Hot-data identification and caching:** Frequently accessed analytical columns can be treated as key columns and prioritized during data-layout optimization. The proposed compact aligned format places important columns efficiently while allowing less frequently accessed columns to occupy remaining storage space, improving effective PIM bandwidth.

III. Query Acceleration Techniques in Modern Data Warehouses

Dynamic Query Processing (DQP) improves query performance by adapting data organization and access structures to observed workload patterns. It uses a Usage Matrix and an Attribute Affinity Matrix to identify frequently accessed attributes and group related data, improving data locality [13]. A cost-based mechanism adjusts partition structures as query patterns change [14]. The framework also divides partitions into smaller units and uses Cuckoo Indexing and Column Imprints to skip irrelevant data during query execution. Evaluation on a 1 GB TPC-H dataset reports approximately 35% lower query execution time than static partitioning, while Column Imprints achieve a tuple-skipping rate of 75%.

A. Query Processing and Acceleration

Fusion focuses on improving analytical query execution in cloud object stores by moving computation closer to the data and reducing unnecessary network transfers. In traditional erasure-coded storage, Parquet column chunks can be divided across multiple storage nodes, creating significant data-reassembly, network, and processing overhead during query execution. Fusion addresses these limitations through file-format-aware coding and fine-grained query pushdown.

- **Analytical query execution:** Executes SQL filtering and projection operations directly on relevant Parquet column chunks at storage nodes.
- **Query processing bottlenecks:** Fragmentation of column chunks requires reassembly across nodes, increasing network latency and CPU overhead; the study observes that network operations can consume around 50% of query time.
- **Query acceleration objectives:** Minimize data movement, network traffic, and query latency by keeping computable units intact and selectively pushing down beneficial query operations.

B. Query Optimization Techniques

Optimizing query execution involves using appropriate execution plans to minimize disk I/O, to avoid table scans that are not necessary, and to minimize the processing overhead. In today's data warehouses, several optimization techniques can be employed to improve query performance [15]:

- **Cost-based query optimization:** The Cost-Based Optimizer (CBO) generates and evaluates alternative execution plans using estimated costs such as disk I/O, then selects the plan with the lowest estimated cost. Updated database statistics help the optimizer make more accurate decisions.
- **Rule-based optimization:** Rule-Based Optimization (RBO) uses predefined transformation rules to improve queries without calculating execution costs. Common rules include predicate pushdown, join elimination, and index-based access.
- **Query rewriting:** Queries can be restructured to eliminate expensive operations. For example, correlated subqueries can be replaced with joins or common table expressions (CTEs) when appropriate to reduce repeated processing.
- **Join and aggregation optimization:** Join operations can significantly increase query cost, particularly on large datasets. Using efficient methods such as hash joins instead of nested loops, along with appropriate indexing, can reduce execution time. Aggregation queries can also benefit from optimized access paths and materialized views.
- **Predicate pushdown:** Filtering conditions should be applied as early as possible so that fewer rows are processed by later operations. Predicate pushdown reduces unnecessary data access and can improve overall query execution performance.

C. Indexing and Data Organization

In today's data warehouses, efficient data retrieval depends on data organization and indexing that reduce unnecessary data access by structuring records and analytical data for efficient retrieval. B-tree and bitmap indexes support selective access, while column-oriented storage enables queries to read only the required attributes and often improves compression. Partitioning limits query processing to relevant data segments, whereas clustering and sorting improve data locality. Zone maps and related data-skipping techniques use block-level metadata to exclude regions that cannot satisfy query predicates, reducing I/O overhead and execution time [16]. These general techniques complement workload-specific frameworks such as DQP, which combine data organization and secondary indexing to improve query efficiency.

D. Parallel and Distributed Query Processing

Query operations and data may be distributed across multiple processing nodes, which can be more efficient when processing large analytical workloads. Parallel queries help to speed up execution of complex queries by running multiple tasks simultaneously. Distributed data processing is achieved through frameworks like Hadoop and Spark, which partition data across the nodes of a cluster, and Massively Parallel Processing (MPP) architectures, which run query parts in parallel on separate nodes [17]. Task and operator parallelism also helps to make more use of the resources by running different stages of the processing simultaneously. Additionally, data shuffling optimization is crucial as it reduces the communication and data movement among nodes, which helps to reduce execution overhead. The cited research proves that MapReduce processing provides better scalability performance, and that Spark's in-memory processing can outperform disk-based Hadoop in large analytical data sets.

E. In-Memory and Hardware-Assisted Acceleration

Modern data warehouses accelerate analytical execution through CPU parallelism, vectorized processing, SIMD instructions, and specialized hardware [18]. Vectorized execution processes batches of values through common operations, reducing instruction overhead, while Single Instruction Multiple Data (SIMD) enables a single instruction to operate on multiple data elements. Graphics Processing Units (GPUs) can accelerate highly parallel analytical

DOI: https://doi.org/xxxxxxxxxxxx

operations, including suitable large-scale data processing and indexing tasks. Hardware-assisted execution can improve throughput for computationally intensive workloads, although its effectiveness depends on parallelism, memory bandwidth, hardware configuration, and data-transfer overhead.

IV. Integrated Data Caching and Query Acceleration in Modern Data Warehouse Systems

The data caching and query acceleration technologies can help modern data warehouse systems provide better analytical performance and help reduce redundant computation [19]. Figure 2 illustrates how these components can operate together within a modern data warehouse. Query-result caching and materialized views reduce repeated computation, while indexing, parallel processing, and hardware-assisted execution accelerate operations that still require processing. Effective integration must also account for data freshness, cache capacity, workload variation, and infrastructure costs. The following subsections examine warehouse architectures, workload-aware optimization, adaptive execution, and end-to-end coordination.

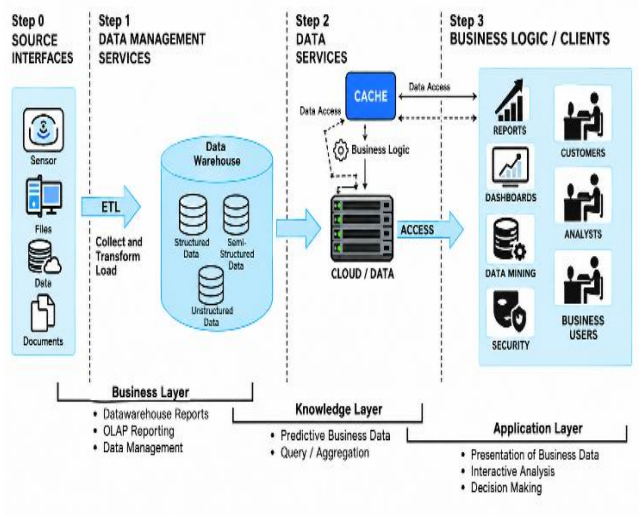


Fig. 2. Data Warehouse Architecture with Cache Mechanism

A. Modern Data Warehouse Architectures

The development of data warehouse systems has resulted in a move from the traditional on-premises infrastructure to cloud and flexible environments [20]. Cloud Data Warehouses (CDWs) provide scalability, elasticity, reliability, and availability, and eliminate the need for organizations to buy and maintain hardware and software infrastructure. Such platforms allow for the flexibility to scale computing resources up or down based on workload needs and facilitate analytical processing of massive amounts of information.

1) Cloud Data Warehouses

Platforms such as Snowflake, Google Big Query, Amazon Redshift, and Azure Synapse support large-scale analytical processing. Their storage, computing, and resource-management capabilities influence query execution and caching strategies.

2) Serverless and Elastic Warehouses

Serverless and elastic execution models allow computing resources to respond to workload demand. Such flexibility can support varying query volumes, although performance and cost depend on resource allocation, execution characteristics, and platform-specific caching mechanisms.

B. Workload-Aware Caching and Query Acceleration

Query access patterns, query execution characteristics, caching potential, and computing resources should be considered to enhance the performance of analytical queries. according to their expected contribution to analytical performance rather than access frequency alone[21]. Decisions can consider query frequency, execution time, data-access patterns, cache capacity, and expected savings from reuse. For example, HPCache uses workload information to prioritize caching opportunities and supports partial-column caching when storing complete datasets would provide limited additional benefit. These strategies aim to allocate cache resources to the data and queries with the greatest expected performance gains.

- **Analytical workload characterization:** Workloads can be characterized using query frequency, data-access patterns, processing intensity, execution time, resource demand, and data dependencies. Data-intensive queries generally benefit more from caching than processing-intensive queries because their performance is more sensitive to input-data access.
- **Workload-aware cache placement:** Cache space is allocated to data that provides the greatest expected query acceleration. HPCache combines query frequency and caching impact and supports partial column caching to avoid wasting memory where additional caching provides little benefit.
- **Query prioritization:** Queries with greater importance or higher expected performance benefit can receive higher caching priority. HPCache can assign different weights to queries and adapt priorities according to recent workload behavior, allowing high-priority queries to receive greater acceleration.
- **Cache Allocation and Prioritization:** Cache space and query priorities can be assigned according to expected performance gains, memory availability, and workload importance. This approach helps direct limited caching resources toward queries that benefit most from data reuse.

C. Adaptive Query Execution and Resource Management

Modern cloud-native databases deploy workload monitoring, predictive models, runtime feedback, and autonomous scaling to adapt their computational resources and query execution strategies as they respond to variations in workload conditions [22]. Unlike static configurations, adaptive approaches use workload monitoring, predictive models, and execution feedback to respond to variations in query complexity, concurrency, and resource availability. Their main objective is to maintain efficient execution while balancing performance requirements and infrastructure costs.

1) Dynamic Resource Allocation

Resource requirements can be predicted from workload characteristics such as query complexity, user concurrency, CPU, memory, and I/O usage. Machine learning models, including LSTM, regression models, and reinforcement learning, can support proactive allocation of resources before performance bottlenecks occur. Resources can then be dynamically adjusted according to predicted demand.

DOI: https://doi.org/xxxxxxxxxxxx

2) Elastic Query Execution

Cloud-native databases can use horizontal and vertical scaling to adapt execution capacity to workload fluctuations. Additional database instances can be added during workload spikes and removed during low-demand periods, while CPU and memory resources can also be adjusted per instance. This elasticity helps maintain query performance while avoiding unnecessary resource consumption.

3) Adaptive Execution Strategies

Query execution can be continuously adjusted using runtime feedback such as query latency, cache-hit ratio, and CPU/memory saturation. Cost-based plan re-evaluation, operator reordering, intelligent index selection, and reinforcement-learning-based optimization can dynamically modify query execution strategies according to current resource availability and workload conditions.

4) Workload-Aware Resource Scheduling

Resource scheduling can consider workload characteristics and query criticality to prioritize important queries and maintain service-level requirements. Predictive workload information can be used to schedule resource-intensive queries during periods of lower utilization, thereby reducing resource contention and improving load balancing. Kubernetes-based orchestration and autonomous scaling policies can further support workload-aware scheduling and dynamic resource management.

D. End-to-End Integration of Caching and Query Acceleration

Modern OLAP workloads have a great advantage in coordinating cache management, data fetching, asynchronously retrieving missing data, and scheduling fine-grained query functions across CPU and GPU resources[23]. By overlapping data retrieval with computation and coordinating execution according to data availability, the framework aims to reduce waiting time and unnecessary data movement.

1) Cache-Assisted Query Processing

The cache manager classifies data batches as Cached, Part Cached, or Uncoached and prioritizes cached batches for execution. Missing columns are fetched asynchronously so that data retrieval can overlap with operator execution.

2) Joint Cache and Query Optimization

ORION jointly considers data availability and operator execution. The cache manager determines an efficient batch-processing order, while the operator manager converts operators into fine-grained functions and schedules them according to available CPU/GPU resources.

3) Coordination Between Storage, Cache, and Execution Layers

ORION connects the OLAP computation layer with remote storage and a distributed cache layer. Required data is retrieved from storage into the cache, converted into columnar batches, and then processed by scheduled functions. Dynamic resource adjustment further coordinates CPU, GPU, and cache resources during execution.

E. Key Challenges and Open Issues

Modern data warehouse systems face several challenges in effectively integrating caching and query acceleration techniques. The major issues include scalability, data freshness, workload changes, cost management, and integration across heterogeneous environments.

- **Scalability:** Increasing data volumes and concurrent analytical queries require caching and query acceleration techniques to operate efficiently across distributed warehouse systems. Maintaining cache efficiency and query performance as the system grows remains a key challenge.
- **Data Consistency and Freshness:** Cached results and materialized views can become outdated when the underlying data changes. Maintaining data freshness while preserving the performance benefits of caching requires effective cache invalidation, refresh, and synchronization mechanisms.
- **Dynamic Workloads:** Data warehouse workloads continuously change in terms of query patterns, complexity, and concurrency. Static caching and optimization strategies may therefore become inefficient, creating a need for adaptive and workload-aware techniques [24].
- **Cost–Performance Trade-offs:** Caching, materialized views, and additional computing resources can improve query performance but also increase storage, memory, and cloud infrastructure costs. Achieving an appropriate balance between performance and resource cost remains an important challenge [25].
- **Heterogeneous Infrastructure and Integration Complexity:** Modern warehouses use diverse storage systems, processing engines, cloud resources, and hardware platforms. Integrating caching, indexing, query optimization, and execution mechanisms across these heterogeneous components can increase system complexity and management overhead.

V. LITERATURE REVIEW

The reviewed studies examine data caching and query acceleration through GPU-based processing, distributed and hybrid caching, reinforcement learning, cost-aware query optimization, and adaptive in-memory caching. These techniques improve query performance, resource utilization, execution efficiency, and cache management across modern analytical environments.

Wang et al. (2026) a Spark-GPU fusion acceleration technology path. The path consists of three key components: first, it integrates the RAPIDS accelerator; second, it designs a GPU-aware partitioning and task co-scheduling strategy; and third, it optimizes the zero-copy data path. Together, these components realize an integrated collaboration of heterogeneous resources. Validation on real-world datasets yields the following results. In real-time aggregation scenarios, the proposed solution improves throughput by a factor of 3.7 over the pure CPU baseline and reduces end-to-end latency by 62%. Compared with the basic GPU solution, GPU utilization rises from 51.7% to 72.3%, representing a relative improvement of 39.8% [26].

C. Gómez-Pantoja, et al. (2025) proposed architecture was evaluated through simulation experiments using real-world query logs (a one-month trace of 1.5 billion queries from a commercial search engine) under multiple cache capacity configurations ranging from 1000 to 100,000 entries and in combination with the Least Recently Used (LRU) and Least Frequently Used (LFU) replacement policies. The

DOI: https://doi.org/xxxxxxxxxxxxx

results show that the proposed architecture consistently achieves higher performance than the baselines, with the largest relative gains in smaller cache configurations and applicability to distributed and hybrid caching environments without fundamental design changes [27].

Sassi and Jaziri (2025) the reinforcement learning and graph-based hybrid query optimizer (GRQO), the first ever to apply reinforcement learning and graph theory for optimizing query execution plans, specifically in join order selection and cardinality estimation. By employing proximal policy optimization for adaptive policy learning and using graph-based schema representations for relational modeling, GRQO effectively traverses the combinatorial optimization space [28].

Lim et al. (2022) a distributed SPARQL query processing scheme for Spark environments that considers communication costs to reduce I/O and data-transfer overhead. SPARQL queries are divided into subqueries based on the WHERE clause, which are then grouped across related nodes using indexing techniques. The approach evaluates the costs of possible query execution paths and selects an efficient path to minimize communication overhead during distributed query processing. This demonstrates the importance of cost-aware query execution and data locality in distributed analytical environments [29].

C. Tang et al. (2024) Petabyte-scale OLAP platforms are increasingly adopting compute-storage decoupling due to the exponential expansion of data, which presents issues including read-heavy I/O traffic, throttling, and fragmented data access. In order to improve data transfer efficiency, this article presents the Alluxio local (edge) cache, an architectural optimization that uses local SSD resources to lower network I/O and API overhead. After three years of use at Uber and Meta, the cache shows efficacy in handling massive business workloads when integrated with OLAP systems like Presto and storage services like HDFS[30].

Anwar Ul Hassan et al. (2022) propose a cache-based mechanism that helps DW in two aspects: the first one is to reduce the execution time by directly accessing records from cache memory, and the second is to save cache memory space by eliminating non-frequent data. target is to fill the cache memory with the most used data. To achieve this goal, an aging-based Least Frequently Used (LFU) algorithm is used by considering the size and frequency of data simultaneously. The priority and expiry age of the data in the cache memory are managed by considering both the size and frequency of data [31].

Table I summarizes recent research on data caching and query acceleration, focusing on GPU-based processing, distributed caching, query optimization, distributed execution, and adaptive cache management. Existing approaches improve performance and resource utilization while highlighting challenges in scalability, adaptability, and workload diversity.

TABLE I. SUMMARY OF THE STUDY ON DATA CACHING AND QUERY ACCELERATION TECHNIQUES

Authors	Research Focus	Approach	Key Findings	Challenges and Limitations	Future Directions
Wang et al. (2026)	GPU-accelerated analytical processing and heterogeneous resource utilization	Integrates RAPIDS, GPU-aware data partitioning, task co-scheduling, and zero-copy data transfer into a Spark-GPU acceleration framework.	Achieved 3.7× higher throughput, 62% lower end-to-end latency, and increased GPU utilization from 51.7% to 72.3% compared with baseline approaches.	Performance depends on GPU availability and efficient coordination between CPU and GPU resources. The evaluation is focused on specific real-time aggregation workloads.	Extend heterogeneous acceleration to more analytical workloads and improve adaptive CPU-GPU resource scheduling for varying workloads.
C. Gómez-Pantoja, et al. (2025)	Distributed and hybrid caching for query-intensive workloads	Evaluates a caching architecture using real-world query traces with different cache capacities and LRU/LFU replacement policies.	The proposed architecture consistently outperformed baseline approaches, with greater improvements under smaller cache capacities and applicability to distributed and hybrid caching.	Evaluation is based on a commercial search-engine workload rather than a data warehouse workload. Cache behavior may vary with different workload characteristics.	Evaluate the approach on analytical and data warehouse workloads and develop adaptive cache policies for changing query patterns.
Sassi and Jaziri (2025)	AI-based query optimization and execution-plan selection	Combines reinforcement learning using PPO with graph-based representations for join-order optimization and cardinality estimation.	GRQO addresses the large query-plan search space through adaptive policy learning and graph-based relational modeling.	RL-based optimization can involve training overhead and depends on the quality of workload and schema representations. Generalization across different database environments requires further validation.	Develop more adaptive learned optimizers and evaluate them across diverse workloads, schemas, and cloud data warehouse environments.
C. Tang et al. (2024)	Improving data access efficiency in petabyte-scale OLAP platforms using local caching.	An Alluxio local (edge) cache using local SSD resources, integrated with Presto and HDFS to reduce network I/O and API overhead.	Three years of deployment at Uber and Meta demonstrated the effectiveness of local caching for large-scale enterprise workloads.	Read-heavy I/O traffic, throttling, and fragmented data access create performance challenges. The study's findings are primarily based on specific enterprise deployments.	Explore adaptive caching policies, improved cache management, and broader evaluation across diverse workloads and data warehouse environments.
Lim et al. (2022)	Distributed query processing and communication-cost reduction	Divides SPARQL queries into subqueries, groups related subqueries using indexes, and selects execution paths based on communication costs.	Reduces data communication and I/O overhead by improving data locality and selecting efficient distributed execution paths.	The approach is designed for SPARQL/RDF graph queries, limiting its direct applicability to relational data warehouses.	Adapt cost-aware distributed query processing to broader analytical and relational workloads and modern cloud data warehouses.
Anwar Ul Hassan	Data warehouse query caching	Uses an aging-based LFU strategy that considers data size,	Reduces query execution time through direct cache access while improving	Cache effectiveness depends on access-frequency patterns and available cache capacity.	Develop workload-aware and adaptive cache replacement mechanisms

et al. (2022)	and cache replacement	access frequency, priority, and expiry age to select frequently used data for caching.	cache-space utilization by removing less frequently used data.	Dynamic workload changes may reduce the effectiveness of static frequency-based decisions.	that incorporate changing query patterns, data freshness, and predictive caching.

VI. Conclusion and Future Work

Data caching and query acceleration are essential for improving analytical performance in modern data warehouse systems. This review covered query result caching, materialized views, in-memory processing, indexing, data partitioning, parallel processing and hardware assisted acceleration. The results show that the combination of these techniques can help to decrease query latency, avoid redundant computation, avoid excessive data transfer, and increase the efficiency of resource utilization. Efficient cache allocation is achieved via workload-aware caching, whereas adaptive query optimization and resource management are intended to meet the evolving analytical requirements. The reviewed studies also highlight the benefits of GPU acceleration, distributed caching and cost-aware execution to achieve better throughput and processing efficiency. But there are issues to address, such as keeping data up-to-date, coping with shifting loads, controlling infrastructure expenses, and orchestrating a heterogeneous stack. In general, the study highlights the potential of combining caching and query acceleration techniques as a viable approach to ensuring scalable, efficient, and economically viable analytical processing in today's data warehouse scenarios.

Future research should develop an integrated, workload-aware framework combining adaptive caching, query optimization, and intelligent resource management. Machine learning can predict query patterns and direct the allocation, replacement and execution planning of the cache. There is a need to conduct further testing for scalability, data freshness, query latency and infrastructure costs with various real-world data warehouse workloads across different cloud environments.

- **Funding:** This research received no external funding.
- **Institutional Review Board Statement:** Not applicable.
- **Informed Consent Statement:** Not applicable.
- **Data Availability Statement:** The data supporting the findings of this study are available from the corresponding author upon reasonable request.
- **Conflicts of Interest:** The author declares no conflict of interest.

How to cite this article:

K. Ahir, "A Review of Data Caching and Query Acceleration Techniques in Modern Data Warehouse Systems," *Journal of Global Research in Multidisciplinary Studies*, vol. 2, no. 10, pp. 22–29, Oct. 2026.Doi.XXXX

References

[1] A. A. Harby and F. Zulkernine, "Data Lakehouse: A survey and experimental study," *Inf. Syst.*, vol. 127, no. 102460, p. 102460, Jan. 2025, doi: 10.1016/j.is.2024.102460.

[2] C. Liu, A. Pavlenko, M. Interlandi, and B. Haynes, "Data formats in analytical DBMSs: performance trade-offs and future directions," *VLDB J.*, vol. 34, no. 3, p. 30, 2025, doi: 10.1007/s00778-025-00911-1.

[3] P. Koukaras, "Data Integration and Storage Strategies in Heterogeneous Analytical Systems: Architectures, Methods, and Interoperability Challenges," *Information*, vol. 16, no. 11, pp. 1–38, 2025, doi: 10.3390/info16110932.

[4] B. Gurumurthy, V. R. Bidarkar, D. Broneske, T. Pionteck, and G. Saake, "Exploiting Shared Sub-Expression and Materialized View Reuse for Multi-Query Optimization," *Inf. Syst. Front.*, vol. 28, no. 3, pp. 941–956, Jun. 2026, doi: 10.1007/s10796-024-10506-w.

[5] Y. Liu, S. Tang, Y. Hui, H. Zhou, and H. Zhang, "Selective Late Materialization in Modern Analytical Databases," *Proc. VLDB Endow.*, vol. 18, no. 11, pp. 4616–4628, Jul. 2025, doi: 10.14778/3749646.3749717.

[6] D. P. Dasepalli, "Secure and Smart Supply Chain Management Systems Using AI-Driven Big Data Analytics and Cloud-Integrated Computing," in *6th International Conference on Computing and Communication Networks (ICCCNet-2026)*, 2026.

[7] R. Ghazali and D. G. Down, "A Systematic Overview of Caching Mechanisms to Improve Hadoop Performance," *Concurr. Comput. Pract. Exp.*, vol. 37, no. 25–26, pp. 1–22, Nov. 2025, doi: 10.1002/cpe.70337.

[8] L. Bindschaedler, "Semantic Caching for OLAP via LLM-Based Query Canonicalization (Extended Version)," Feb. 2026. doi: 10.48550/arXiv.2602.19811.

[9] J. You *et al.*, "SOC : A Succinct Adaptive Semantic OLAP Caching," *Data Sci. Eng.*, vol. 10, no. 4, pp. 665–680, 2025, doi: 10.1007/s41019-025-00290-1.

[10] X. Hu and Z. Miao, "Benchmarking the Full Pipeline of Materialized-View-Based Query Rewriting," *arXiv*, vol. 19, no. 11, pp. 01–22, Jul. 2026, doi: 10.14778/3836663.3836724.

[11] S. Kilaru, V. S. R. Yarlagadda, and M. B. Nagaraja, "AI-Augmented Block-Sketch Hybrid Compression: Enabling Semantic Query Processing on Compressed Data," in *2026 IEEE 16th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA: IEEE, 2026, pp. 0966–0972, Jan. doi: 10.1109/CCWC67433.2026.11393750.

[12] Y. Zhao, F. Liu, and G. Chen, "PUSHtap : PIM-based In-Memory HTAP with Unified Data Storage Format," *Proc. 30th ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst. Vol. 3 (ASPLOS '25), March 30-April 3, 2025, Rotterdam, Netherlands*, vol. 1, no. 1, pp. 01–16, 2025, doi: 10.1145/3676642.3736120.

[13] K. Gandhi, M. Bhise, and M. Kamdar, "DQP: Dynamic Query Processing for Varying Workload," in *2026 IEEE Region 10 Symposium (TENSYMP)*, Penang, Malaysia: IEEE, Jun. 2026, pp. 1–5. doi: 10.1109/TENSYMP68909.2026.11704059.

[14] S. Park, M. Jeong, and H. Han, "CCA: Cost-Capacity-Aware Caching for In-Memory Data Analytics Frameworks," *Sensors*, vol. 21, no. 7, p. 2321, Mar. 2021, doi: 10.3390/s21072321.

[15] J. H. Lubis, S. Handayani, H. Mawengkang, and F. M. A. Napitupulu, "Performance Optimization of ERD Designs Using Cost-Based Optimization for Large-Scale Query Processing," *J. Tek. Inform.*, vol. 6, no. 3, pp. 1457–1468, Jun. 2025, doi: 10.52436/1.jutif.2025.6.3.4523.

[16] A. Abbas and K. Ahmad, "An Integrated Indexing Approach to Improve Query Processing," in *2021 International Conference on Electrical Engineering and Informatics (ICEEI)*, IEEE, Oct. 2021, pp. 1–6. doi: 10.1109/ICEEI52609.2021.9611104.

[17] E. Azhir, M. Hosseinzadeh, F. Khan, and A. Mosavi, "Performance Evaluation of Query Plan Recommendation with Apache Hadoop and Apache Spark," *Mathematics*, vol. 10, no. 19, p. 3517, Sep. 2022, doi: 10.3390/math10193517.

[18] M. Abbasi, M. V Bernardo, P. Váz, J. Silva, and P. Martins, "Revisiting Database Indexing for Parallel and Accelerated Computing: A Comprehensive Study and Novel Approaches," *Information*, vol. 15, no. 8, p. 429, Jul. 2024, doi: 10.3390/info15080429.

[19] P. Jinka, "Data Warehouse Optimization Techniques: A Survey Of Query Processing, Storage, And Performance Enhancement Methods,"," *World J. Adv. Eng. Technol. Sci.*, vol. 20, no. 2, pp. 216–226, Aug. 2026, doi: 10.30574/wjaets.2026.20.2.0408.

[20] J. Oliveira e Sá, R. Gonçalves, and C. Kaldeich, "Benchmark of Market Cloud Data Warehouse Technologies," *Procedia Comput. Sci.*, vol. 239, pp. 1212–1219, 2024, doi: 10.1016/j.procs.2024.06.289.

[21] H. Nicholson, P. Chrysogelos, and A. Ailamaki, "HPCache: memory-efficient OLAP through proportional caching revisited," *VLDB J.*, vol. 33, no. 6, pp. 1775–1791, 2023, doi: 10.1007/s00778-023-00828-7.

[22] P. Sundaramoorthy, M. Parikh, S. R. Keshireddy, A. K. Penumatcha, and K. Jayabalan, "Adaptive Resource Management in Cloud-Native Databases: A Study of Autonomous Scaling and Query Optimization," in *2025 10th International Conference on Communication and Electronics Systems (ICCES)*, IEEE, Oct. 2025, pp. 1506–1512. doi: 10.1109/ICCES67310.2025.11336962.

[23] Z. Tong *et al.*, "ORION: Optimizing OLAP Query Execution with Proactive Caching and Separate Operators," in *Proceedings of the 39th ACM International Conference on Supercomputing*, in ICS '25. New York, NY, USA, NY, USA: ACM, Jun. 2025, pp. 868–883. doi: 10.1145/3721145.3725752.

[24] M. Hentschel, J. Dees, F. Funke, M. Heimel, and I. Oukid, "Building a Data Management System for the Cloud: Lessons Learned and Future Directions," *Datenbank-Spektrum*, vol. 25, no. 1, pp. 17–28, 2025, doi: 10.1007/s13222-025-00494-9.

[25] H. Zhang, Y. Liu, and J. Yan, "Cost-Intelligent Data Analytics in the Cloud," Aug. 2023. doi: 10.48550/arXiv.2308.09569.

[26] X. Wang, C. Xie, J. Ding, B. Xu, J. Li, and W. Huang, "Performance Optimization of Distributed Data Processing in Centralized Control System Based on Spark and GPU Collaboration," *Information*, vol. 17, no. 7, p. 625, Jun. 2026, doi: 10.3390/info17070625.

[27] C. Gómez-Pantoja, D. Baeza-Rocha, and A. Inostrosa-Psijas, "A Behavior-Aware Caching Architecture for Web Applications Using Static, Dynamic, and Burst Segmentation," *Futur. Internet*, vol. 17, no. 9, p. 429, Sep. 2025, doi: 10.3390/fi17090429.

[28] N. Sassi and W. Jaziri, "Efficient AI-Driven Query Optimization in Large-Scale Databases: A Reinforcement Learning and Graph-Based Approach," *Mathematics*, vol. 13, no. 11, p. 1700, May 2025, doi: 10.3390/math13111700.

[29] J. Lim, B. Kim, H. Lee, D. Choi, K. Bok, and J. Yoo, "An Efficient Distributed SPARQL Query Processing Scheme Considering Communication Costs in Spark Environments," *Appl. Sci.*, vol. 12, no. 1, 2022, doi: 10.3390/app12010122.

[30] C. Tang *et al.*, "Data Caching for Enterprise-Grade Petabyte-Scale OLAP," Jun. 2024. doi: 10.48550/arXiv.2406.05962.

[31] C. A. U. Hassan *et al.*, "Optimizing the Performance of Data Warehouse by Query Cache Mechanism," *IEEE Access*, vol. 10, pp. 13472–13480, 2022, doi: 10.1109/ACCESS.2022.3148131.