Efficient Storage and Query Optimization for Large-Scale Data Sets in Distributed Architectures

Yonggang Zhao[1,*]
[1]School of Information Engineering, Henan Logistics Vocational College, Zhengzhou City, Henan Province, 450012, China
Corresponding author: Yonggang Zhao (e-mail: zyg19820201@126.com).

ABSTRACT With the rapid development of the big data industry, data volume across various industries has exploded, and large-scale datasets at PB and EB levels have become mainstream objects for data processing. Relying on core theories of distributed storage and query, this paper constructs an integrated collaborative optimization system covering storage, query and caching. Storage efficiency is improved by designing an adaptive dynamic sharding strategy and intelligent multi-replica placement policy, building a tiered storage architecture for hot and cold data, and optimizing storage encoding and compression mechanisms. Query overhead is reduced by reconstructing query execution plans based on cost models, pushing down operators, and optimizing cross-node transmission. A collaborative global-local indexing framework and a multi-level caching linkage mechanism are established to further boost query response performance. A standardized distributed experimental cluster is deployed, and multi-dimensional comparative experiments are conducted to verify the performance of the proposed optimization scheme. Experimental results demonstrate that the proposed optimization solution can effectively cut storage redundancy overhead, improve cluster resource utilization, drastically reduce query latency for large-scale data, and raise concurrent throughput. It can provide theoretical support and engineering practice references for distributed system optimization in industrial big data, internet massive data processing and other scenarios.

INDEX TERMS Distributed Architecture, Large-Scale Datasets, Storage Optimization, Query Optimization, Cache Collaboration

I. INTRODUCTION

A. RESEARCH BACKGROUND AND RESEARCH SIGNIFICANCE

1) Challenges in Big Data Storage and Query

With the popularization of the Internet of Things, cloud computing and artificial intelligence technologies, data collection dimensions in government affairs, finance, industry, the Internet and other fields have been continuously enriched, and data volume has rapidly expanded from TB to PB and EB levels, covering structured, semi-structured and unstructured multi-modal data.

The continuous accumulation of massive data imposes higher requirements on storage systems in terms of capacity scalability and read-write stability, as well as on query systems in terms of real-time performance and high concurrency.

Traditional centralized data architectures rely on single-node hardware performance to support businesses, suffering from fixed storage capacity, limited computing power, high expansion costs, single-point failure risks and other drawbacks. When handling large-scale datasets, such architectures frequently encounter storage overflow, query stalling, task timeout and other failures, failing to meet the operational demands of modern big data businesses.

2) Application Status of Distributed Architectures

Distributed clusters support parallel storage and computing across multiple nodes, featuring horizontal scalability, high availability and strong fault tolerance, and have become the mainstream architecture for big data processing. Frameworks including Hadoop, Spark and TiDB are widely adopted in massive data storage, offline analysis, real-time query and other business scenarios, serving as the infrastructure for big data processing.

Nevertheless, most commercial and open-source distributed systems are general-purpose architectures without customized optimization tailored to the access characteristics and storage properties of large-scale datasets, leading to prominent performance bottlenecks under ultra-large data volumes and high-concurrency query scenarios.

3) Core Value of Efficient Storage and Query Optimization for System Performance

Storage and query performance directly determine the resource utilization and business response speed of distributed clusters. Carrying out integrated optimization research can reduce system redundancy, balance load and shorten query latency without additional hardware investment, delivering high engineering practical value.

B. DOMESTIC AND OVERSEAS RESEARCH STATUS

1) Research Progress and Typical System Comparison of Distributed Storage Systems

Overseas research on distributed storage technologies started earlier, forming a mature technical system. Google successively proposed distributed storage systems such as GFS, Bigtable and Spanner, realizing distributed storage and strong consistency management of massive data with excellent stability and scalability. However, these architectures are complex and incur extremely high deployment costs, making them unsuitable for small and medium-sized business scenarios. HDFS and HBase launched by the Apache open-source community have become industry mainstream due to open-source accessibility and complete ecosystems, adopting fixed block splitting and multi-replica storage mechanisms to guarantee data reliability.

Domestic research focuses on lightweight optimization and domestic localization adaptation. Domestic distributed databases including OceanBase and TiDB optimize replica mechanisms and storage structures to better fit domestic business scenarios, yet they still suffer from deficiencies in dynamic load adaptation.

Table 1 compares the characteristics of mainstream distributed storage systems.

2) Research Status of Distributed Query Optimization Technologies

Overseas mainstream computing engines Spark and Flink optimize query execution plans through query parsing, syntax optimization and operator pushdown, and implement parallel task scheduling based on Directed Acyclic Graphs (DAGs), greatly improving batch query efficiency. In terms of cost model optimization, overseas scholars have proposed multi-dimensional cost evaluation models incorporating data scale, node computing power and network bandwidth to enable intelligent selection of query execution plans.

Domestic research mostly concentrates on scenario-specific optimization, conducting special optimizations for hot queries, multi-table join queries and massive data aggregation queries, yielding abundant achievements in single-point technologies such as index optimization and query caching. However, a full-process query optimization system remains absent.

Synthesizing domestic and overseas research status, existing studies generally suffer from single-dimensional optimization, poor dynamic adaptability, low resource utilization and insufficient versatility, making them inapplicable to ultra-large-scale, high-concurrency massive data scenarios.

C. RESEARCH CONTENT AND SIGNIFICANCE

1) Research Content

Centered on the core objective of storage and query performance optimization for large-scale datasets under distributed architectures, this paper conducts systematic research:

a) Sort out core theories and key technologies related to distributed storage, distributed query processing and performance optimization, and analyze core bottlenecks of existing systems;

b) Construct a storage optimization system, designing adaptive dynamic sharding, tiered storage for hot and cold data and intelligent encoding compression strategies to resolve low storage efficiency and load imbalance;

c) Establish a collaborative mechanism integrating query optimization and index caching, lowering query latency through query plan reconstruction, index architecture optimization and caching upgrade;

d) Deploy an experimental cluster to verify the effectiveness of the proposed optimization scheme from multiple dimensions including storage performance, query performance and resource utilization, and summarize research achievements and future optimization directions.

2) Theoretical and Practical Significance

Theoretically, this paper constructs an integrated collaborative optimization framework covering storage, query and caching, making up for the limitations of traditional single-module optimization and further improving the theoretical system of optimization for large-scale distributed data processing.

The proposed adaptive optimization strategies require no high hardware investment and can adapt to mainstream distributed big data frameworks. They effectively enhance cluster storage utilization and concurrent query performance while reducing system operation and maintenance costs, and can be widely applied to engineering scenarios such as massive data analysis and data governance, possessing sound practical application value.

II. RELEVANT THEORETICAL AND TECHNICAL FOUNDATIONS

A. DISTRIBUTED STORAGE ARCHITECTURE THEORY

1) Distributed File System Architecture and Data Distribution Principles

Distributed file systems adopt a master-slave cluster architecture, consisting of management nodes and data nodes as core components.

The core principle of data distribution lies in sharding storage of large files: oversized files are split into multiple data blocks of fixed size and scattered across different DataNodes. A multi-replica replication mechanism stores data replicas on separate nodes to balance parallel storage

TABLE 1. Comparison of Characteristics of Mainstream Distributed Storage Systems

Storage System	Storage Mode	Sharding Method	Replica Mechanism	Applicable Scenarios	Core Deficiencies
HDFS	File Storage	Static Fixed Sharding	Fixed Three-Replica Storage	Offline storage of massive unstructured data	Fails to dynamically adapt to data access characteristics
HBase	Columnar Storage	Static Region Sharding	Synchronous Multi-Replica Storage	High-frequency read-write of structured data	Prone to hot data aggregation and unbalanced load
Spanner	Distributed Relational Storage	Dynamic Sharding	Multi-Region Replicas	Globally distributed high-reliability businesses	Complex architecture, high deployment cost
TiDB	NewSQL Hybrid Storage	Range Sharding	Configurable Replica Count	Online transaction processing of structured data	No differentiated optimization for hot and cold data storage

and data fault tolerance, effectively preventing data loss caused by single-node failures.

2) Comparative Analysis of NoSQL and NewSQL Storage Models

NoSQL databases abandon the strong transaction characteristics of traditional relational databases, adopting columnar, key-value and document storage architectures with high availability, high scalability and low latency. They fit high-frequency read-write scenarios of massive unstructured and semi-structured data but lack support for complex transactions and multi-table join queries.

NewSQL databases integrate the transaction consistency of traditional SQL databases and the horizontal scalability of distributed architectures, perfectly adapting to transaction processing and complex query scenarios of massive structured data, and have become the mainstream choice for enterprise-level big data businesses.

Table 2 details the comparison between the two storage models.

3) Data Sharding, Replica Management and Consistency Protocols

Data sharding is the core technology of distributed storage, divided into static sharding and dynamic sharding. Massive data is split and evenly allocated to cluster nodes to realize parallel processing of read-write tasks.

Replica management serves as the key to guarantee high availability of distributed systems. Each data block is assigned 1 – 3 replicas stored on separate cluster nodes. Upon primary node failure, the system automatically switches to replica nodes to undertake business, avoiding data loss and service interruption.

Consistency protocols are mainly used to guarantee synchronous consistency of multi-replica data. Mainstream protocols including Paxos and Raft resolve asynchronous data and data inconsistency across distributed nodes through leader election, log synchronization and data verification mechanisms, ensuring cluster data reliability.

B. DISTRIBUTED QUERY PROCESSING TECHNOLOGIES

1) Query Parsing and Execution Plan Generation Mechanism

The distributed query processing workflow consists of five core stages: query parsing, syntax verification, logical plan generation, physical plan optimization and task execution.

After receiving user SQL statements, the system first completes lexical and syntactic parsing to verify statement validity. An initial logical query plan is then generated to sort out fields, operators and data tables required for queries. Combined with cluster node status and data distribution, the system optimizes and generates the optimal physical execution plan, determining sharding, scheduling and execution nodes for query tasks. Finally, split subtasks are distributed to each node for parallel execution, and query results are aggregated and returned to users.

2) Distributed Computing Models Such as MapReduce and DAG

MapReduce is a classic offline distributed computing model adopting a two-stage architecture of Map sharding processing and Reduce aggregation output. It realizes batch query and computation of massive data with outstanding stability yet suffers from low efficiency for iterative computation.

The DAG computing model represents a new generation of stream computing architecture. Query tasks are decomposed into multiple interdependent operator nodes executed in parallel following topological structures, supporting iterative computation and real-time streaming queries. Compared with MapReduce, it drastically reduces task startup and execution latency and has become the mainstream model for real-time distributed queries.

3) Acceleration Technologies Including Columnar Storage and Vectorized Execution

Columnar storage splits data tables by columns. Queries only load required field data without reading redundant full-row data, significantly cutting disk I/O overhead and suiting scenarios of massive data aggregation queries and field filtering queries.

Vectorized execution abandons the traditional single-row data processing mode and executes query operators in batches with data vectors as units, reducing system overhead from function calls and loop traversal and greatly improving CPU utilization and query execution efficiency. It serves as a core acceleration technology for large-scale data queries.

III. DISTRIBUTED STORAGE OPTIMIZATION STRATEGIES FOR LARGE-SCALE DATASETS

A. ADAPTIVE DATA SHARDING AND LAYOUT OPTIMIZATION

TABLE 2. Comparison of NoSQL and NewSQL Storage Models

Storage Model	Transaction Capability	Scalability	Query Complexity	Read-Write Latency	Applicable Data Type
NoSQL	Weak/No Transaction	High Horizontal Scalability	Adapted to Simple Queries	Low Latency	Unstructured, Semi-Structured Data
NewSQL	Strong Transaction Consistency	Elastic Horizontal Scalability	Complex Queries, Multi-Table Joins	Medium-Low Latency	Massive Structured Data

1) Dynamic Sharding Algorithm Based on Data Access Characteristics

To address the defect that traditional static sharding fails to adapt to dynamic access scenarios, this paper designs a dynamic sharding algorithm based on data heat and access frequency.

Abandoning fixed-size sharding, the algorithm collects real-time characteristic parameters including data access frequency, access time and read-write heat to build a data access characteristic evaluation model.

For hot small-scale data with high access frequency, fine-grained small sharding is adopted to avoid excessive node load caused by oversized shards. For cold low-access data and batch static data, large shard merging storage is applied to reduce metadata overhead of shards, enabling dynamic adaptive adjustment of shard size according to data characteristics.

2) Hot Data Identification and Balanced Distribution Strategy

A multi-dimensional hot data identification model is constructed, quantifying data heat values with core indicators including access frequency, read-write times and query weights to classify data into three tiers: hot data, warm data and cold data.

For identified hot data, the fixed storage node binding mechanism is broken. Cross-node shard migration and uniform allocation disperse hot data across multiple cluster nodes to prevent single-node overload from concentrated hot data accumulation and realize balanced cluster load.

3) Intelligent Multi-Replica Placement and Fault Tolerance Mechanism

Traditional fixed three-replica placement strategies lead to excessive storage redundancy and resource waste. This paper proposes an adaptive replica placement mechanism.

Core hot data retains three replicas to guarantee high availability; ordinary warm data adopts dual replicas to balance performance and overhead; low-frequency cold data uses erasure coding instead of multiple replicas to reduce storage redundancy.

Meanwhile, replica placement rules are optimized: different replicas of the same data are stored across different racks and physical servers to avoid overall data loss caused by rack failures and enhance system fault tolerance.

B. TIERED STORAGE ARCHITECTURE FOR HOT AND COLD DATA

1) Data Heat Evaluation Model and Determination Mechanism

Combining the time decay law of data access and access frequency characteristics, a quantitative heat evaluation model is built with set heat threshold intervals.

Real-time heat values are calculated by counting data access times and read-write frequencies over the past 7 and 30 days combined with time decay coefficients, automatically dividing data into hot, warm and cold tiers without manual intervention.

2) Collaborative Management Scheme for Multi-Level Storage Media

A three-tier storage media architecture of "Memory – SSD High-Speed Disk – HDD Mechanical Disk" is established to realize differentiated storage for hot and cold data.

The memory layer stores ultra-hot query cache data to guarantee millisecond-level response;

SSD high-speed disks store hot and warm data to balance read-write speed and storage cost;

HDD mechanical disks store massive cold data to drastically reduce hardware storage costs.

Different tiers of storage media are precisely matched to data of varying heat levels to achieve the optimal balance between storage performance and cost.

3) Design of Automatic Data Migration and Scheduling Algorithm

A heat decay-based automatic migration scheduling algorithm is designed to monitor real-time changes in data heat and regularly update hot-cold data tiers.

When hot data heat decays to the warm data threshold, it is automatically migrated to the SSD storage layer;

When warm data continuously cools down to the cold data threshold, it is migrated to the HDD storage layer;

When cold data is frequently accessed and its heat rises, it is recalled to the high-speed storage layer in real time.

The entire migration process runs silently in the background without affecting front-end business operations, realizing dynamic adaptive scheduling of data storage locations.

C. STORAGE COMPRESSION AND ENCODING OPTIMIZATION

1) Adaptive Compression Algorithm Selection Oriented to Query Workloads

An adaptive compression algorithm selection mechanism is designed:

TABLE 3. Parameters and Applicable Scenarios of Multi-Level Storage Media

Storage Tier	Storage Medium	Read-Write Speed	Storage Cost	Applicable Data Types
Primary Cache Layer	Memory	Nanosecond Level	High	Ultra-Hot Cache Data, Hot Indexes
Secondary High-Speed Storage Layer	SSD Disk	Microsecond Level	Medium	Hot Data, Warm Data
Tertiary Mass Storage Layer	HDD Disk	Millisecond Level	Low	Low-Frequency Cold Data, Archived Data

GZIP high-compression algorithm is adopted for massive text log data to improve compression ratio and cut storage overhead;

LZ4 fast compression algorithm is used for numerical structured query data to reduce compression and decompression time and boost query speed;

Lightweight compression algorithms are applied to small-size high-frequency access data such as indexes and metadata to avoid decompression overhead.

2) Performance Trade-Off between Erasure Coding and Replica Strategies

A dynamic trade-off mechanism is constructed: multi-replica strategies are prioritized for core business data with high availability requirements to guarantee stable read-write performance. For massive archived cold data, erasure coding replaces multiple replicas, reducing storage redundancy from 200% to below 50% while ensuring data recoverability and greatly lifting storage utilization.

3) Joint Optimization Model for Storage Overhead and Query Performance

A bidirectional joint optimization model covering storage overhead and query performance is established. Storage redundancy rate and disk occupancy are taken as storage overhead indicators, while query latency and I/O throughput serve as performance indicators. A weighted algorithm is applied to obtain the optimal solution for dual objectives.

Compression levels, replica counts and storage tiers are dynamically adjusted to avoid increased query decompression latency caused by excessive compression and resource waste from excessive redundancy, realizing dynamic balance between storage cost and query performance.

IV. DISTRIBUTED QUERY OPTIMIZATION AND COLLABORATIVE INDEX-CACHING MECHANISM

A. DISTRIBUTED QUERY PLAN OPTIMIZATION

1) Query Rewriting and Optimization Based on Cost Model

A multi-dimensional query cost evaluation model is built, quantifying resource overhead of different query execution plans by comprehensively considering node computing power, network bandwidth, data distribution and operator complexity.

For redundant operators, invalid scans and repeated calculations in initial query plans, automatic query rewriting is performed to eliminate invalid execution steps and preferentially select low-overhead, high-parallelism execution plans, cutting query resource consumption from the source.

2) Multi-Table Join Order and Operator Pushdown Strategy

The multi-table join sorting rule is optimized following the core principle of "small tables drive large tables, small datasets drive large datasets". Filtering is prioritized to reduce intermediate result volume.

A full-scale operator pushdown strategy is implemented, pushing filtering, screening, aggregation and other basic operators down to data storage nodes for execution. Only processed result data are aggregated instead of transmitting raw data across nodes, drastically lowering network I/O overhead.

3) Minimization Method for Cross-Node Data Transmission

Based on computation migration theory, task scheduling mechanisms are optimized to strictly follow the principle of "local data computation priority". Query tasks are preferentially scheduled to nodes storing target data.

For query tasks requiring inevitable cross-node association, local pre-aggregation and shard-local aggregation strategies are adopted. Global aggregation is performed only after local data aggregation on each node to minimize cross-node data transmission volume and mitigate the impact of network latency on query performance.

B. DISTRIBUTED INDEX STRUCTURE DESIGN

1) Collaborative Architecture of Global and Local Indexes

Local indexes build field indexes for local data on each node, fitting single-node local query scenarios with fast query speed and low maintenance cost. Global indexes coordinate data across all cluster nodes, recording global data shard distribution and index information to adapt to cross-node association queries and full-data queries.

The two types of indexes work collaboratively to automatically match the optimal index type according to query scope, balancing local query efficiency and global query accuracy.

2) Secondary Index Construction and Incremental Maintenance Mechanism

For massive structured data query scenarios, secondary indexes for high-frequency query fields are constructed to break the limitation of relying solely on primary key indexes.

Meanwhile, an incremental index maintenance mechanism is designed: only index data of corresponding shards are synchronized and updated upon data insertion, update or deletion, eliminating full index reconstruction and drasti-

cally reducing index maintenance overhead to avoid system performance jitter during large-scale data updates.

3) Index Partitioning and Load Balancing Strategy

Global indexes are partitioned and stored following data sharding rules to evenly distribute index data across cluster nodes and prevent index query bottlenecks from concentrated index storage.

Index query load on each node is monitored in real time, and index partition distribution is dynamically adjusted to realize load balancing of index query tasks and guarantee system stability under high-concurrency index query scenarios.

C. MULTI-LEVEL COLLABORATIVE CACHING OPTIMIZATION

1) Layered Design of Query Result Cache and Data Block Cache

A two-tier cache layered architecture is established:

Tier 1: Query result cache, storing complete results of high-frequency queries to directly match repeated query requests and achieve sub-second response;

Tier 2: Data block cache, storing high-frequency raw data blocks and index data to support new queries without full matching.

Layered caching realizes precise caching of data at different granularities, avoiding cache redundancy and cache missing problems.

2) Cache Replacement Algorithm Based on Access Patterns

Traditional LRU cache replacement algorithms suffer from elimination of aged hot data. This paper optimizes the cache replacement strategy, calculating retention priority of cached data by combining multi-dimensional parameters including data access frequency, access duration and query weight.

High-weight and high-heat cached data are prioritized for retention, while low-frequency and expired cached data are eliminated to lift cache hit ratio and reduce system overhead from cache replacement.

3) Cache Preheating and Invalidation Consistency Maintenance

To address low cache hit ratio during cold startup, a scheduled cache preheating mechanism is designed. Based on historical access logs, high-frequency query data and indexes are automatically loaded into caches during business off-peak hours to guarantee high cache hit ratios during business peak hours.

Meanwhile, a cache invalidation synchronization mechanism is established: when underlying data is updated, corresponding cached data is cleared and updated in real time to avoid query errors caused by inconsistency between cached data and underlying data.

V. EXPERIMENTAL VERIFICATION AND PERFORMANCE ANALYSIS

A. EXPERIMENTAL ENVIRONMENT AND EVALUATION SCHEME

1) Distributed Cluster Deployment and Hardware Configuration

A distributed test cluster consisting of 1 management node and 8 data nodes is deployed with uniform hardware configuration for all nodes. The operating system is CentOS7.9, the distributed framework adopts Hadoop3.2 + Spark3.3, and the database uses TiDB6.5.

2) Test Datasets and Benchmark Workload Design

The open-source big data test dataset TPC-H is adopted to generate structured datasets of 100GB, 500GB and 1TB, simulating small-scale, medium-scale and large-scale massive data scenarios.

The benchmark Workload adopts the 22 standard complex TPC-H query statements covering mainstream query types including single-field filtering, multi-table association, aggregation statistics and sorting grouping. Simultaneously, 10, 50 and 100 concurrent users are simulated to test system performance under different concurrency scenarios.

3) Multi-Dimensional Quantitative Performance Evaluation Indicators

A multi-dimensional quantitative evaluation indicator system is established to comprehensively assess the performance of the proposed optimization scheme:

Storage performance indicators: storage compression ratio, disk occupancy, data writing time, node load balance degree;

Query performance indicators: average single-query response time, concurrent query throughput, query success rate;

Resource utilization indicators: CPU utilization, memory utilization, network I/O utilization.

The native unoptimized distributed system serves as the control group, and the proposed optimization scheme acts as the experimental group for comparative experiments.

B. VERIFICATION OF STORAGE OPTIMIZATION EFFECTS

1) Comparative Analysis of Storage Overhead and Compression Ratio

Under the 1TB dataset scenario, storage occupancy and compression effects of the native system and optimized system are compared. The native system adopts fixed three-replica and unified storage strategies with high storage redundancy. The proposed optimization scheme drastically cuts storage overhead through adaptive compression, hot-cold tiered storage and collaborative erasure coding and replica strategies.

Experimental results show that the overall storage compression ratio of the optimized system rises by 28.6%, disk storage occupancy drops by 32.3%, and storage resource utilization is significantly improved.

TABLE 4. Hardware and Software Configuration Parameters of Experimental Cluster

Configuration Item	Specific Parameters
Node Quantity	1 Management Node + 8 Data Nodes
CPU Configuration	8-core, 16 threads, 2.5GHz Main Frequency
Memory Configuration	16GB DDR4
Disk Configuration	500GB SSD + 2TB HDD
Network Bandwidth	1000M Gigabit Network Card
System Version	CentOS7.9
Core Frameworks	Hadoop3.2, Spark3.3, TiDB6.5

2) Data Loading and Writing Performance Test

Batch data loading and writing time under different data volumes are tested. Results reveal that under 100GB, 500GB and 1TB datasets, data writing speed of the optimized system increases by 12.4%, 18.7% and 25.2% respectively.

The core reason lies in dynamic sharding optimization eliminating shard congestion, while hot-cold tiered storage allocates high-speed disks for hot write data to reduce I/O waiting latency. The optimization effect becomes more prominent as data volume expands.

3) System Scalability and Fault Tolerance Verification

Cluster horizontal expansion performance is tested by gradually adding data nodes. The native system tends to suffer unbalanced load after expansion, while the proposed optimization scheme achieves a 41.5% improvement in node load balance degree via intelligent replica placement and dynamic shard scheduling, delivering stronger stability for cluster expansion.

Single-node and single-rack failures are simulated to verify fault tolerance. The optimized system completes replica switching and data recovery rapidly, cutting fault recovery time by 35.8% and demonstrating markedly enhanced fault tolerance.

C. EXPERIMENTAL ANALYSIS OF QUERY PERFORMANCE

1) Comparison of Single-Query Response Time

Typical complex TPC-H query statements are selected to test average single-query response time under a 1TB dataset with zero concurrency. Table 5 presents performance comparison between the control group and experimental group.

Response time under all query scenarios is drastically shortened after optimization, with the most significant improvement observed in multi-table association and aggregation queries. This benefits from query plan optimization, operator pushdown and collaborative index-caching mechanisms, which drastically reduce cross-node transmission and redundant computation overhead.

2) Concurrent Query Throughput Test

Three tiers of concurrent query scenarios (10, 50 and 100 concurrent users) are set to test the number of query tasks processed by the cluster per second.

Experimental results indicate that throughput rises by 22.3% under 10 concurrency, 30.6% under 50 concurrency and 38.9% under 100 high concurrency.

The optimization effect becomes more prominent under high concurrency, proving that collaborative index-caching mechanisms and load balancing strategies can effectively cope with high concurrent query pressure and avoid system performance degradation.

3) Performance under Different Data Scales

Full-data query tests are conducted on 100GB, 500GB and 1TB datasets. As data volume expands, query latency of the native system rises exponentially with severe performance degradation.

The proposed optimization scheme maintains stable performance even under ultra-large data volumes. Under the 1TB dataset scenario, average query latency drops by 52.7% compared with the native system, demonstrating excellent adaptability and stability for big data.

D. COMPREHENSIVE PERFORMANCE EVALUATION

1) Comparison of End-to-End Task Execution Efficiency

Three typical big data tasks including batch data import, full-data statistics and complex multi-dimensional analysis are selected to test overall end-to-end execution time.

After optimization, execution efficiency of the three task types rises by 24.5%, 47.2% and 51.3% respectively, drastically improving overall business processing efficiency. This verifies that the integrated optimization scheme comprehensively elevates the overall operational capacity of distributed systems.

2) Analysis of System Resource Utilization

Experimental monitoring results show that after optimization, average cluster CPU utilization rises from 48.2% to 72.5%, memory utilization increases from 53.6% to 81.2%, and effective network I/O utilization improves by 36.8%.

Refined resource scheduling, load balancing and elimination of redundant overhead resolve the problems of idle resources and low utilization in traditional distributed systems, realizing efficient utilization of cluster computing and storage resources.

TABLE 5. Performance Comparison of Response Time Across Different Query Types

Query Type	Native System Response Time (ms)	Optimized System Response Time (ms)	Performance Improvement Rate
Single-Table Filter Query	426	218	48.8%
Multi-Table Association Query	1285	572	55.5%
Data Aggregation Statistical Query	968	405	58.2%
Sorting and Grouping Query	753	326	56.7%

3) Comprehensive Evaluation of Optimization Strategy Effectiveness

Synthesizing multiple groups of experimental results, the proposed storage optimization, query optimization and collaborative index-caching optimization strategies all deliver remarkable effectiveness.

At the storage layer, cost reduction and efficiency improvement are realized, drastically cutting storage redundancy and hardware overhead;

At the query layer, latency is significantly reduced and concurrent capacity lifted;

At the resource layer, refined scheduling maximizes cluster computing and storage value.

The overall optimization effect outperforms single-module optimization schemes, effectively meeting the demand for efficient storage and query of large-scale datasets.

VI. CONCLUSION AND PROSPECT

Aiming at pain points of distributed architectures for large-scale datasets including high storage redundancy, unbalanced node load and excessive query latency, this paper constructs an integrated optimization system integrating storage optimization, query optimization and collaborative index caching. Multiple groups of comparative experiments verify that the proposed scheme can effectively improve cluster storage utilization, resource utilization and data query performance, and significantly mitigate operational shortcomings of traditional distributed systems.

Nevertheless, the proposed scheme still has certain application limitations: its adaptability to streaming storage and query of ultra-large unstructured files and lightweight deployment scenarios on small clusters requires further improvement, and the experimental scenarios are relatively idealized.

Future research can further integrate heterogeneous hardware characteristics, artificial intelligence adaptive tuning technologies and cloud-native elastic architectures to continuously optimize intelligent scheduling, dynamic adaptability and elastic expansion capabilities of distributed systems, building a more universal, intelligent and high-performance storage and query system for large-scale data.

REFERENCES

[1] S. Patil and S. Sangam, "An Exhaustive Survey of Big Data Storage Reduction Techniques," *Cureus J. Comput. Sci.*, vol. 2, no. 1, p. 3518, 2025, doi: 10.7759/S44389-025-03518-3.

[2] S. Guan, C. Zhang, Y. Wang, et al., "Hadoop-based secure storage solution for big data in cloud computing environment," *Digit. Commun. Netw.*, vol. 10, no. 1, pp. 227–236, 2024, doi: 10.1016/J.DCAN.2023.01.014.

[3] G. Jifu, H. Chunlin, H. Jinliang, "A Scalable Computing Resources System for Remote Sensing Big Data Processing Using GeoPySpark Based on Spark on K8s," *Remote Sens.*, vol. 14, no. 3, p. 521, 2022, doi: 10.3390/RS14030521.

[4] B. J. Kim, "Verification of scalability and compatibility of TiDB for DBMS migration," *J. Digit. Contents Soc.*, 2022, doi: 10.9728/dcs.2022.23.11.2299.

[5] S. Huijun and R. Ruonan, "Scalable distributed RDFS reasoning using MapReduce and Bigtable," Shanghai Jiao Tong Univ., China, 2013, doi: 10.1117/12.2010731.

[6] A. Sale and N. Enas, "Survey on a Google File System (GFS)," *Int. J. Comput. Appl.*, vol. 181, no. 28, pp. 9–16, 2018, doi: 10.5120/ijca2018918084.

[7] J. C. C, H. Peter, H. Wilson, et al., "Spanner: Google's Globally Distributed Database," *ACM Trans. Comput. Syst.*, vol. 31, no. 3, pp. 1–22, 2013, doi: 10.1145/2518037.2491245.

[8] H. Fei, Y. Chaowei, J. Yongyao, et al., "A hierarchical indexing strategy for optimizing Apache Spark with HDFS to efficiently query big geospatial raster data," *Int. J. Digit. Earth*, vol. 13, no. 3, pp. 410–428, 2020, doi: 10.1080/17538947.2018.1523957.

[9] H. Su, J. Li, L. Guo, et al., "Massive Data HBase Storage Method for Electronic Archive Management," *Int. J. Netw. Manage.*, vol. 35, no. 1, p. e2308, 2024, doi: 10.1002/NEM.2308.

[10] K. Yan, J. Jia, L. Lv, et al., "OceanBase Database Health Status Prediction System Based on the HPSA-LSTM Model," *Acad. J. Comput. Inf. Sci.*, vol. 8, no. 8, 2025, doi: 10.25236/AJCIS.2025.080809.

[11] C. Yang, D. Qingyang, W. Dan, et al., "TIDB: a comprehensive database of trained immunity," *Database*, 2021, doi: 10.1093/DATABASE/BAAB041.

[12] R. M. Kaseb, Haytamy, et al., "Distributed query optimization strategies for cloud environment," *J. Data Inf. Manage.*, pp. 1–9, 2021, doi: 10.1007/S42488-021-00057-Z.

[13] L. Jongtao, K. Byounghoon, L. Hyeonbyeong, et al., "An Efficient Distributed SPARQL Query Processing Scheme Considering Communication Costs in Spark Environments," *Appl. Sci.*, vol. 12, no. 1, p. 122, 2021, doi: 10.3390/APP12010122.

[14] Y. Du, Z. Ding, Z. Cai, et al., "Join query optimization in distributed database based on multi-source mating selection evolutionary algorithm," *Cluster Comput.*, vol. 28, no. 5, p. 281, 2025, doi: 10.1007/S10586-024-04905-6.

[15] K. Fan, "Research on distributed query optimization technology based on index," M.S. thesis, Univ. Electron. Sci. Technol. China, Chengdu, China, 2024, doi: 10.27005/d.cnki.gdzku.2024.005185.

[16] Z. Aoujil, M. Hanine, Z. Baybih, et al., "NoSQL Vis: A unified visualization and exploration platform for heterogeneous NoSQL datastores," *SoftwareX*, vol. 35, p. 102778, 2026, doi: 10.1016/J.SOFTX.2026.102778.

[17] R. R. E. F. Ferreira and N. D. R. Fidalgo, "A Performance Analysis of Hybrid and Columnar Cloud Databases for Efficient Schema Design in Distributed Data Warehouse as a Service," *Data*, vol. 9, no. 8, p. 99, 2024, doi: 10.3390/DATA9080099.