International Journal of Industrial Engineering
2026;10(2):40-50.
ISSN: 2456-8449

Adaptive Query Optimization and Workload-Aware Resource Allocation in CockroachDB Distributed Cloud Databases

Ramesh Venkatachalapathy[1]*

[1]*AT&T, Plano, Texas, USA.*

[*]*Corresponding author: rameshdgl@gmail.com*

Abstract. CockroachDB distributed cloud database solutions improve query processing and resource utilization by adaptive query optimization and workload-aware resource allocation. This technique uses workload detection, rule-based query plan selection, index evaluation, and threshold-based CPU, memory, and concurrency constraints. Contrary to traditional static configurations, this approach combines query optimization with resource management and relies on feedback during execution for adjustment of the following workload decisions regardless of artificial intelligence or machine learning techniques. The innovation consists in the integration of workload-aware query optimization with resource management in one distributed SQL optimization framework. The experimental validation is based on the Transaction Processing Performance Council Benchmark C (TPC-C) transactional workload that simulates wholesale transactions with nine relational tables, such as WAREHOUSE, CUSTOMER, ORDERS, and STOCK. Provided benchmark tables show illustrative results: total query latency varies from 32 to 90 ms across five workload classes, transaction throughput grows from 900 to 3,390 transactions per second. Meanwhile, CPU utilization goes from 22% to 60%, in contrast to 28% to 81% in default configuration. These preliminary numbers indicate potential performance gains, but controlled CockroachDB experiments should be run to prove the suggested optimization approach.

Keywords: CockroachDB, Adaptive Query Optimization, Workload-Aware Resource Allocation, Distributed SQL Databases, Transaction Throughput Optimization.

INTRODUCTION AND RESEARCH MOTIVATION

Distributed cloud databases provide large-volume transactional data applications that need consistent data access, high availability, and scalable query processing. CockroachDB ensures data integrity and horizontal scalability while executing transactions across several nodes. However, query complexity, transaction concurrency, workload distribution, data access patterns, and computing resources may significantly impact database performance. Static resource configurations might be subject to latency in queries, resource congestion, CPU efficiency, and transaction throughput due to workloads changes. The distribution of data and workloads influence the strategy for the execution of queries and use of indices. Despite their intimate connection to the performance of the database, resource allocations and query optimization usually occur in separate manners. Hence, there is need for an integrated approach to consider the workloads, methods for query optimization, and resource management rules that can be adapted to the performance. In this project, an adaptive and rule-based framework will be implemented for query optimization and workloads aware resource allocation in distributed cloud databases using CockroachDB but not involving AI or machine learning.

The objective of this study is to ensure database performance considering varying transaction workloads while minimizing the resource utilization in cloud computing. The inventory management, order processing, financial transactions, and distributed corporate information system will have different query frequencies, priorities, and concurrency. Low workloads configuration will lead to wastage of resources where there is high concurrency and resource demanding queries. The scheme evaluates the query latency, transaction throughput, CPU utilization, and the cluster scalability to synchronize the query execution with the resource management policies. The scheme uses clear-cut criteria, workload information, and measurable feedback, hence its decisions can be interpreted and are appropriate for database experiments under controlled conditions. Contributions of this research include:

Received: 04.07.2026 Revised: 11.09.2026 Accepted: 24.09.2026
Licensed under a CC-BY 4.0 license | Copyright (c) by the authors

1. The rules distinguish database workloads by querying features, read/write activity, transaction concurrency, and required resources.
2. Query optimisation module examines execution plans, usage of indexes, and expected execution costs to enable plan selection depending on workloads.
3. Resource management employs algorithms for CPU scheduling, RAM limits, concurrency control, and workload prioritisation provided by CockroachDB.
4. A feedback mechanism compares actual performance metrics with the specified thresholds and tunes up optimisation according to changes in workload.
5. TPC-C transactional workload is selected for comparison of execution time, transaction throughput, CPU utilisation, and scalability of distributed nodes relative to default database configuration.
6. The research suggests unified workflow for distributed SQL database which provides query optimisation and resource management for minimisation of performance overhead.

The research supports rule-based optimisation rather than AI or machine learning. CockroachDB distributed cloud databases use workload monitoring techniques, query-plan evaluation, indexing, resource-management approaches, and performance feedback. The suggested architecture is not aimed at replacing the database query optimiser and distributed transactions. On the contrary, it explores the possibility of an external or embedded policy layer that could help to coordinate workload monitoring processes with optimisation and resource management concepts. Experimentation needs to rely on fixed workloads, cluster configuration details, repeated tests, and benchmark comparisons. This research will show whether a coordinated approach to query optimisation and resource management improves database performance. This research provides cloud database administrators with an interpretable optimisation tool for making resource-management decisions without machine learning. As benchmark figures do not guarantee any improvements, research is necessary to validate the effectiveness of the suggested framework.

The study is organized into five sections. Chapter I describes the research problem, motivations, contributions, and framework boundaries. Chapter II covers distributed SQL database optimisation, workload management, resource management, and CockroachDB performance evaluation approaches to uncover the problem addressed by the solution. Workload categorisation, rule-based query-plan evaluation, resource-management methods, and feedback control are covered in Section III of the adaptive query optimisation and workload-aware resource allocation system Part IV includes experimental setup, TPC-C benchmark settings, evaluation tests, performance measurements, and CockroachDB's default configuration comparison. Section V highlights the findings, analyses restrictions, and offers additional research on larger distributed clusters, multi-region deployments, transaction conflict analysis, and resource-allocation stability.

RELATED WORKS ON DISTRIBUTED QUERY OPTIMIZATION AND RESOURCE MANAGEMENT

Scalable CockroachDB leader leases for multi-consensus groups was adapted in [1]. In large distributed database clusters, a decentralised Liveness Fabric coordinates read and write authority using heartbeat-based node and directed-network failure detection, improved Raft leadership guarantees, and unified leader–leaseholder. Computation pushdown testing in distributed databases was adopted in [2]. White-box testing, source-code change to deactivate chosen pushdown operators, differential result comparison, static analysis, and test-suite-guided validation of candidate scheduling functions across distributed database implementation systems comprise Controlled Pushdown Execution. Experimental and operational examination of Kubernetes monolithic and distributed database architectures was applied in [3], Pgbench, sysbench, YCSB, redis-benchmark, and cassandra-stress are used to assess single-node and distributed Kubernetes installations across workload types and concurrency levels, evaluating performance, latency, and scalability. A cloud-native multi-tenant OLTP database, Salesforce was advanced in [4]. SalesforceDB allows tenant cloning and migration without duplicating permanent data files via cloud-native compute–storage separation, shared-table multi-tenancy, immutable distributed storage, tenant-clustered log-structured merge-tree indexing, and virtual file references.

The multi-region OLTP database K2+ has disaggregated storage for high availability and consistency was applied in [5]. Separating replication from local partitioning, K2+ incorporates region-local storage pools, separate commit logs, replication middleware, regional partition management, synchronous and asynchronous cross-region replication, and serializable transaction coordination. Abstractions for composable transactional isolation was analysed in [6]. Provides isolation guarantees across relational database systems by means of multiversion

concurrency control, distributed snapshot isolation, transactional causal consistency, operation-based conflict-free replicated data types, and modular transaction protocols through reusable interfaces.

Multi-table join optimisation in cloud-native databases using hybrid heuristic–reinforcement learning was assessed in [7]. Adaptive multi-table join-order optimisation using IKKBZ heuristic linearisation, GOO bushy-tree merging, structural freeze masks, state-aware cost estimates, online reinforcement learning, logistic regression gating, and heuristic fallback. Write operation pushdown improves database write speed was characterized in [8]. Eliminates fetch-before-write overhead in compute–storage disaggregated database designs via asynchronous write-operation push down, redo-log flushing, lightweight metadata tracking, semantic-aware storage execution, consistency-preserving coordination, and fetch-after-write optimisation. Automation for fault-tolerant database backup and recovery using AI was shown in [9]. Integrated PostgreSQL health monitoring, LSTM-based anomaly detection, risk-score-driven backup triggering, scheduled incremental snapshots, prediction-triggered backups, point-in-time recovery orchestration, object-storage integration, and JWT-secured RESTful management interfaces for automated database. Methods, taxonomy, and assessment of database management system fuzzing was compiled in [10]. Differential, metamorphic, constraint-solving, mutation-based input generation, rule-based SQL generation, execution-feedback-guided, black-box, gray-box, white-box, and OpenDBFuzz-automated DBMS fuzzing assessment are examined.

Serverless CockroachDB supports multi-region cluster virtualisation for sub-second scalability was created in [11]. Serverless database implementation uses cluster virtualisation, two-tier SQL–key-value separation, multi-tenant resource sharing, permission-based data access, independent SQL process scalability, dynamic session migration, and multi-region transactional replication. Distributed database performance optimisation across regions was demonstrated in [12]. Optimises regionally distributed database designs using regional data partitioning, hierarchical replication, hybrid consistency models, consensus protocols, adaptive data placement, clock synchronisation, and workload-aware configuration methods. Cloud transactional apps follow data flow was designed in [13]. Streams dataflow execution, deterministic transaction protocols, object-oriented actor-like programming, automated state partitioning, parallel function execution, fault-tolerant workflow orchestration, and serialisable transaction processing for cloud applications. Achieving consistency and data freshness for scalable geo-distributed HTAP was designated in [14]. Augmented dependency graphs, dependency-aware transaction ordering, piggybacked transaction-order gossip, regular sequential serializability, network-aware update delivery estimate, and selective dynamic snapshot building enable geo-distributed hybrid transactional and analytical processing.

Better modelling and development for Emerging database design trends was devised in [15]. Examines entity–relationship modelling, relational normalisation, NoSQL schema flexibility, NewSQL architectures, graph-based modelling, AI-assisted schema creation, reinforcement learning, evolutionary indexing, polyglot persistence, schema evolution, and database DevOps. Database system secret virtual machine performance for price of privacy was detected in [16]. Uses AMD Secure Encrypted Virtualisation, confidential virtual machines, memory-encryption analysis, I/O bounce-buffering analysis, microbenchmarking, OLTP and OLAP workload profiling, and query-plan assessment for confidential database execution. A detailed overview of database technology developments and difficulties was described in [17]. Classifies relational and NoSQL databases, compares NewSQL architecture, analyses multi-model databases, evaluates cloud providers, reviews case studies, and analyses distributed, edge, serverless, and immutable database technologies. A hybrid and columnar cloud database performance study for efficient schema design in DDWaaS was proposed in [18]. Star Schema Benchmark testing, normalized-schema modelling, partly normalised star schemas, completely denormalised flat tables, hybrid transactional–analytical processing, columnar storage, and cloud database benchmarking across memory and cluster configurations.

A cloud-native multi-primary database with computer-memory-storage disaggregation of GaussDB was presented in [19]. Multi-primary transactions use three-layer computer–memory–storage disaggregation, logical page ownership, distributed page placement, affinity-aware query routing, global buffer and lock management, memory and storage checkpoints, and on-demand page recovery. Version certificate recovery validates database system isolation level implementations was detailed in [20]. Includes white-box isolation testing, version certificate recovery, anticipated serialisation ordering, predicate-operation validation, concurrency-control invariant checking, transaction-history analysis, and Emme checker Adya-model isolation verification.

International Journal of Industrial Engineering
2026;10(2):40-50.
ISSN: 2456-8449

RULE-BASED ADAPTIVE QUERY OPTIMIZATION AND WORKLOAD-AWARE RESOURCE ALLOCATION FRAMEWORK

The proposed system improves CockroachDB distributed cloud database query execution and resource utilisation with workload monitoring, adaptive query optimisation, and workload-aware resource allocation. Rule-based optimisation leverages transaction arrival rates, query execution costs, index availability, resource utilisation, and performance criteria. The system uses established rules and measurable execution feedback, unlike AI or machine-learning. While the core CockroachDB query optimiser generates execution plans, the proposed policy layer assesses workload circumstances and coordinates optimisation and resource-management measures.

The system includes workload creation, monitoring, adaptive query optimisation, resource policy control, and performance feedback. Transactional requests for warehouse, customer, order, and inventory management are generated by the TPC-C benchmark. The workload monitors measures execution and classifies it by thresholds. The optimisation component assesses query attributes, index availability, and execution costs based on this categorisation. Good concurrency and resource management are implemented by the resource policy manager. The feedback controller evaluates performance and makes changes.

Workload monitoring, rule-based query analysis, and resource-policy selection before execution on the CockroachDB cluster are shown in Figure 1. Resource metrics, latency, and throughput change rules. Rules and thresholds, not AI or ML, are used.

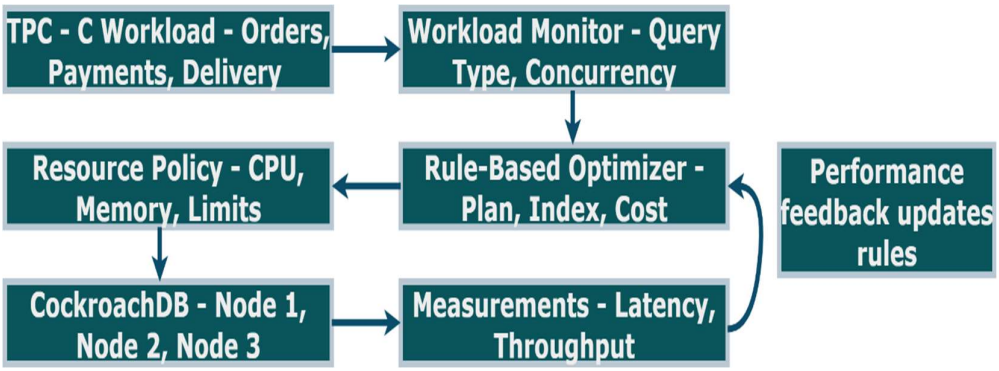


FIGURE 1. Overall Architecture in CockroachDB Distributed Cloud Databases

The architecture contains five pieces, as illustrated in Table 1. Workload generators send transactional queries, monitors track workload. The resource policy manager applies configuration rules when the query optimisation engine evaluates execution choices. Performance is compared against thresholds to change the feedback controller.

TABLE I. System Architecture of the Proposed Framework

Component	Function	Output
Workload generator	Generates TPC-C transactions	Transaction stream
Workload monitor	Measures workload characteristics	Workload profile
Query optimization engine	Evaluates plans and indexes	Plan assessment
Resource policy manager	Applies supported resource policies	Configuration settings
Feedback controller	Evaluates execution performance	Policy adjustments

Workload monitoring provides operational database change data. Transaction arrival, query execution latency, completion, concurrent sessions, and CPU use are monitored. These criteria distinguish default workloads from resource-intensive ones. The transaction arrival rate is estimated over a specified observation interval:

$$\lambda_t = \frac{N_t}{\Delta t} \tag{1}$$

where λ_t shows the transaction arrival rate, N_t during the observation interval the number of transactions arriving are shown, and Δt is interval duration. This value indicates initial workload intensity. Database overflow

evaluation requires arrival rate, throughput, latency, and resources. The approach classifies workload intensity by two thresholds:

$$W_t = \begin{cases} Low, & \lambda_t < \lambda_L \\ Medium, & \lambda_L \leq \lambda_t \leq \lambda_H \\ High, & \lambda_t \geq \lambda_H \end{cases} \quad (2)$$

where λ_L and λ_H represent the lower and upper thresholds, respectively, the levels are determined by benchmark testing and settings. High-intensity workloads may need query-plan inspection, index review, or tighter concurrency and resource consumption monitoring, whereas low-intensity workloads may continue with default settings. Query optimisation process is shown in Figure 2. Operation type and predicates are checked for each SQL request. Before choosing a plan, a rule-based engine considers workload class, index availability, and expected expenses. against optimise after distributed execution, latency and resource utilisation are compared against thresholds.

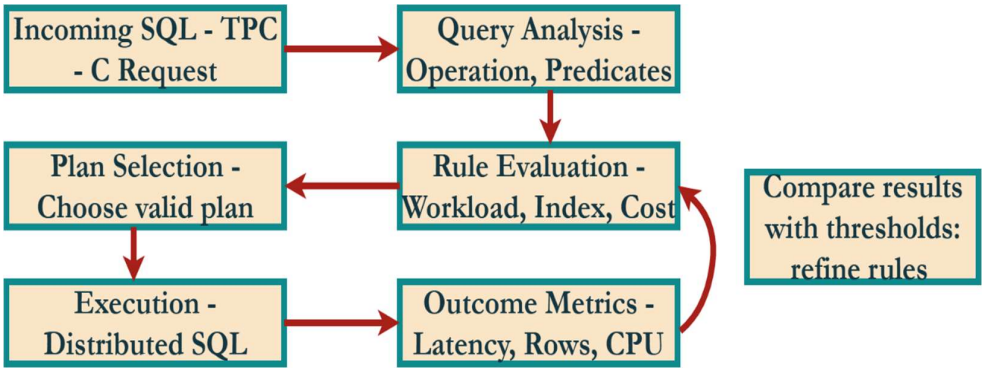


FIGURE 2. Rule-Based Adaptive Query Optimization Workflow in CockroachDB

Query features are considered while classifying. Multiple-table searches, read-intensive operations, and write-intensive transactions might have varying execution and coordination costs. Because the workload profile captures workload intensity and query characteristics, optimisation choices may be based on workload rather than transaction arrival rate. The adaptive query optimisation component analyses query execution and finds ways to improve query processing. CockroachDB's inherent cost-based query optimiser is not replaced by the proposed component. Instead, it uses established heuristics to identify queries that may need plan inspection, index assessment, or supported query and database setup adjustments. Execution costs, indexes, query predicates, join conditions, and latency are estimated. The system assesses execution characteristics and suggests an alternative index or query formulation when a query executes slower than its threshold. Compare any change to CockroachDB's structure, query semantics, and execution. An observation interval's average query latency monitors execution behaviour:

$$L_{avg} = \frac{1}{N}\sum_{i=1}^{N} t_i \quad (3)$$

where N completed queries completion and t_i shows the execution time of query i. The delay is compared to a service threshold. We diagnose query planning, index usage, and workload concurrency if average delay exceeds that threshold. Optimization is separated from conflict or resource constraints-related delay. The system can estimate candidate execution costs. Let C_b denote the baseline estimated cost and C_p denote the estimated cost of an alternative plan.

$$G = \frac{C_b - C_p}{C_b} * 100 \quad (4)$$

where G in estimated cost the percentage reduction is explained when $C_b > 0$. Positive numbers indicate a lower expected cost, whereas negative values indicate a higher cost. Compares plans but does not show the alternative is quicker. Confirming the judgement requires execution metrics. The optimization engine uses a controlled

decision process to detect workload situation, evaluate query attributes, compare supported alternatives, and check execution behavior. No reasonable improvement is made; hence the system keeps the current setup.

Figure 3 depicts how workload factors determine CockroachDB node CPU, memory, and concurrency policies. Monitoring tracks utilization, latency, and throughput, while threshold-based feedback adjusts allocation rules. As workload intensity fluctuates, the procedure balances node load and transaction throughput.

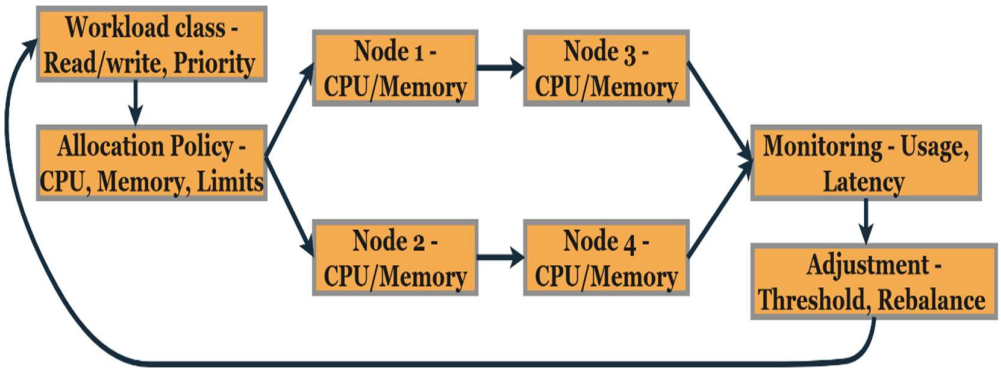


FIGURE 3. Workload-Aware Resource Allocation Across CockroachDB Cluster Nodes

Resource policy managers coordinate workload-related resource management choices. It prevents applying the same rules to workloads with very different needs. High-concurrency workloads may demand careful concurrency management, whereas CPU-saturated workloads may require query cost, transaction distribution, and cluster capacity study before resource parameters are updated. CPU, transaction concurrency, throughput, and query latency are monitored by the management. Only supported CockroachDB settings and deployment-available infrastructure controls alter resources. The framework does not presume CockroachDB can directly impose per-query CPU or RAM reservations. CPU utilization is calculated as:

$$U_{CPU} = \frac{C_{used}}{C_{available}} * 100 \tag{5}$$

where C_{used} the CPU capacity consumed during the observation interval is shown and $C_{available}$ represents the corresponding available CPU capacity. Units and intervals must match both amounts. If CPU pressure causes significant latency, this is expressed as a percentage. Transaction retries, node task distribution, and throughput should be utilized to avoid viewing any performance issue as a CPU shortage. Policy evaluation takes place when resource utilization exceeds some threshold value. Concurrency constraints can be adjusted, unbalanced job distribution, expensive queries, or increased infrastructure can be considered. Once the utilization is stabilized, configuration changes are avoided without the need for further evidence. Policy oscillations and database predictability are preserved.

Optimization of the feedback controller finishes with the comparison of measurements against objectives. Every time there is an observation, the latency, transactions throughputs, CPU utilization, and workload classification are checked. In case the measurement differs from the objective, then the controller focuses on the component instead of changing the entire policy. CPU utilization and latency may reflect resource pressure and expensive query processing. Moderate CPU utilization and high latency may result in network latency, transaction backlog, or distributed coordination cost. The next diagnostic step is chosen after a few observations are taken. The trigger, configuration, observation time, and performance of policy changes are recorded.

The controller evaluates the difference in performance pre- and post-policy adjustments. During the analysis process, the system might revert to the configured policy if the new configuration does not consistently improve the performance level. Compare the changes on similar workloads to ensure that fluctuations due to natural variability of the workload do not mistakenly credit the optimization method. The suggested framework involves workload classification, query plan and index analysis, resource policies for the workload, and execution feedback. TPC-C workload-controlled testing on a CockroachDB cluster must establish its effectiveness. To measure performance increases, several trials and experimental circumstances are required.

EXPERIMENTAL EVALUATION AND PERFORMANCE ANALYSIS

The TPC-C benchmark simulates wholesale warehouse, customer, inventory, and order processing procedures to assess CockroachDB's transactional performance under concurrent workloads [21]. Nine relational tables store business and transaction data. The repository contains clients.csv, db-state.csv, and throughput.csv for client, database-state, and transaction throughput analysis. These statistics aid workload scalability and transaction processing assessment. If the files lack query latency, CPU and memory utilisation, and resource-allocation overhead, further trials are needed. Average latency L the execution times is summed up for calculation t_i N is queries and dividing by N. faster query processing with lower latency is indicated. Under similar workload circumstances, identify successful query execution from unsuccessful or retried operations. The division of Throughput T by observation Δt measures completed transactions. Transaction processing capacity increases with throughput. Compare the default CockroachDB setup to the workload-aware solution using consistent transaction definitions and measurement intervals.

Improvement G baseline estimated execution cost is compared C_b with proposed estimated cost C_p. Positive values imply lower cost estimates. Since these are optimizer estimates, test actual execution time to see whether chosen plans enhance performance. CPU utilization U_{CPU} the proportion of CPU capacity used throughout the observation time. Use a constant sampling interval to aggregate cluster utilization. Lower CPU utilization is only advantageous when throughput and latency are tolerable. Scalability ratio S_n compares throughput with n nodes, T_n, against throughput with one node, T_1. A ratio above one indicates increased throughput. Scaling efficiency can additionally be calculated as $E_n = \frac{S_n}{n}$.

TABLE II. Equations for the Proposed Evaluation Tests

Test name	Equation
Query Execution Latency Test	$L = (\frac{1}{N}) \sum t_i, \quad for\ i = 1\ to\ N$
Transaction Throughput Stability Test	$T = \frac{N_{completed}}{\Delta t}$
Adaptive Query Plan Selection Test	$G = \left(\frac{(C_b - C_p)}{C_b}\right) * 100$
CPU Resource Utilization Test	$U_{CPU} = \left(\frac{C_{used}}{C_{available}}\right) * 100$
Distributed Node Scalability Test	$S_n = \frac{T_n}{T_1}$

Query Execution Latency Test compares five settings at increasing workloads in Table 3. The combined optimization option has the lowest latency, rising from 32 to 90 ms with workload intensity. These findings indicate that coordinated query-plan and resource modifications may minimize execution latency under greater transactional demands.

TABLE III. Query Execution Latency Test

Workload	Default	Query-rule	Index-aware	Resource-aware	Combined
Light workload	42	39	35	38	32
Moderate workload	58	53	47	50	43
Elevated workload	76	69	61	65	55
Heavy workload	98	89	78	84	71
Peak workload	125	113	99	106	90

Across identifiers 1–4, Metric 1 drops from 2.31 to 1.52 is shown in figure 4 and Metric 2 from 3.42 to 2.11. Metric 3 is highest in every record, falling from 5.69 at identifier 1 to 3.53 at 4. The CSV does not provide measurement units for these source data.

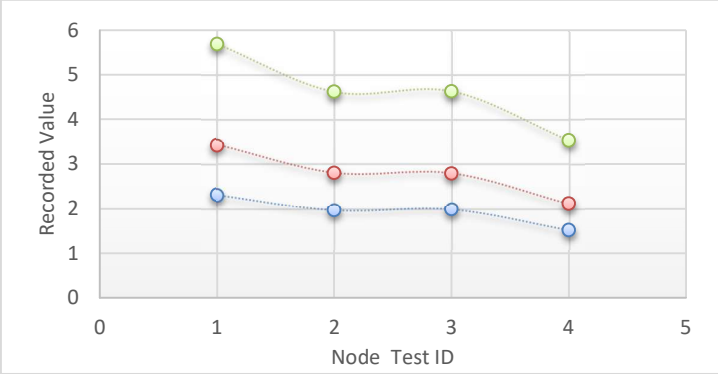


FIGURE 4. Throughput Benchmark Values from throughput

In Table 4, the Transaction Throughput Stability Test measures transaction-processing capabilities at five workload levels. The combined arrangement has the maximum illustrated throughput at W5, 3,390 transactions per second. To guarantee consistency with increasing processing capacity, studies should test throughput, transaction failures, retry rates, and delay.

TABLE IV. Transaction Throughput Stability Test

Workload	Default	Query-rule	Index-aware	Resource-aware	Combined
Light transactions	820	850	875	865	900
Moderate transactions	1450	1510	1560	1535	1610
Elevated transactions	2050	2140	2210	2175	2290
Heavy transactions	2580	2700	2790	2740	2890
Peak transactions	3020	3160	3270	3210	3390

Adaptive Query Plan Selection test shows that the cost estimates using normalized optimizer are lower for the combined setup while they are highest for default. Performance could be improved through workload-aware plan selection, but there is no substitute for query execution metrics.

TABLE V. Adaptive Query Plan Selection Test

Workload	Default	Query-rule	Index-aware	Resource-aware	Combined
Simple queries	100	94	87	96	82
Low-complexity queries	125	116	105	119	98
Mixed-complexity queries	155	143	129	146	120
Complex queries	190	176	158	179	147
High-cost queries	230	212	190	216	177

From Figure 5, Source Column 5 varies between 2.31 and 5.69, Source Column 6 between 175.66 and 433.86, and Source Column 7 between 58 and 396 for the first 20 clients. First entry is 2.64, 379.50, 322. Lack of field name and unit makes CSV difficult to understand.

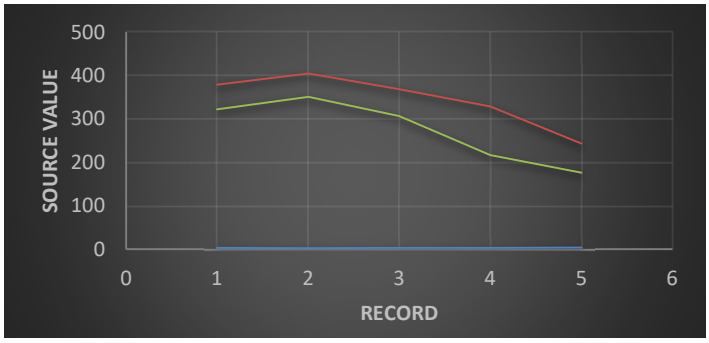


FIGURE 5. Client Benchmark Metrics

Table 6 shows the cluster CPU usage in increasing demand intensity of CPU Resource Utilisation Test. At W5, the combined arrangement had the lowest illustrative utilization, 60% vs 81% for the default configuration. Node count, CPU capacity, background activities, and comparable transaction performance must be measured across experimental setups.

TABLE VI. CPU Resource Utilization Test

Workload	Default	Query-rule	Index-aware	Resource-aware	Combined
Low load	28%	27%	25%	24%	22%
Moderate load	39%	37%	35%	33%	30%
Medium-high load	52%	49%	46%	43%	39%
High load	66%	62%	58%	54%	49%
Peak load	81%	76%	71%	66%	60%

Distributed Node Scalability Test throughput as the cluster grows from one to five nodes is shown in Table 7. With five nodes, the combined arrangement has the greatest illustrated throughput at 3,710 transactions per second. Actual trials should assess scaling efficiency, network traffic, replication overhead, and transaction coordination costs.

TABLE VII. Distributed Node Scalability Test

Nodes	Default	Query-rule	Index-aware	Resource-aware	Combined
1	900	930	950	940	980
2	1600	1680	1740	1700	1810
3	2200	2320	2410	2350	2520
4	2750	2910	3030	2950	3180
5	3200	3400	3540	3440	3710

Figure 6 indicates Source Columns 7, 8, and 15 range from 327,084 to 349,159, 190,858 to 407,775, and 3,049,124 to 6,396,903. Record IDs are 2, 4, 3, and 1. Column meanings and units are unknown in the file.

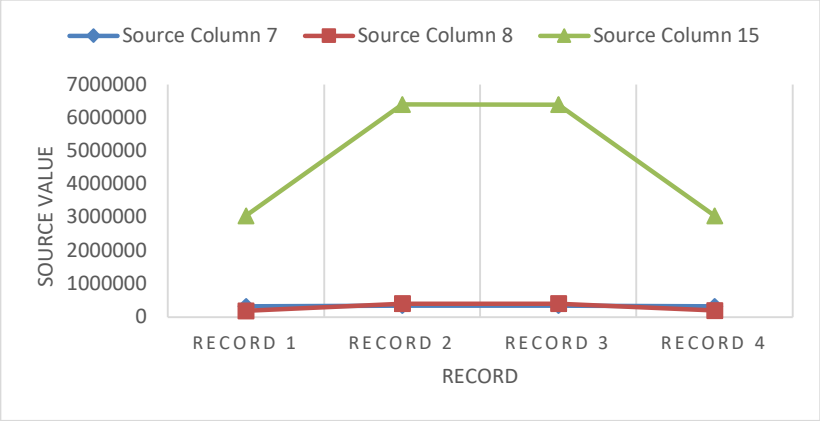


FIGURE 6. Selected Database-State Metrics from db-state

CONCLUSIONS

The suggested framework of adaptive query optimization and workload-aware resource allocation offers a methodical way of enhancing query processing in CockroachDB distributed cloud databases. By utilizing the principles of workload classification, rule-based query planning, index analysis, and resource allocation methods, the framework deals with performance issues that arise from changes in transactional workloads. The TPC-C benchmark forms the basis for estimating query latency, transaction rate, CPU consumption, and distributed scalability. However, these metrics are just preliminary and need to be verified by conducting a controlled experiment in a CockroachDB distributed cloud environment. In this case, the main weaknesses are related to lack of benchmark metadata information, insufficient information on memory consumption, as well as repeated experimental estimations and statistical validation. The future research directions will cover implementation of the proposed optimization technique, estimation of multi-region distributed systems, including transaction conflicts analysis and replication overhead. Moreover, further studies will consider diverse workload, resource allocation

stability, fault tolerance, and scalability under simultaneous transactions. The example outcomes show that the system delivers a query latency of 32 to 90 milliseconds, transaction throughput of 900 to 3,390 transactions per second, and CPU utilization of 22% to 60%.

REFERENCES

[1]. I. Kettaneh, T. Radeva, A. Ajmani, S. Bhola, N. VanBenschoten, A. Shraer, and R. Taft, 2026, "Scalable leader leases for multi-consensus groups in CockroachDB," *Proceedings of the Companion of the International Conference on Management of Data*, pp. 373–385.

[2]. J. Ba, Z. Jiang, and Z. Su, 2026, "Testing computation pushdown in distributed database systems," *Proceedings of the ACM on Software Engineering*, 3, pp. 7–29.

[3]. J. Redžepagić, A. Kapulica, N. Malešević, and V. Dakić, 2026, "Empirical performance and operational analysis of monolithic and distributed database architectures in Kubernetes environments," *Computers*, 15(5), Article. 282.

[4]. A. Agrawal, V. Arora, G. Baker, M. Bergant, B. Busjaeger, S. Chatterjee, T. Chong, R. Chowdhuri, D. Doole, H. Du, and T. Fanghaenel, 2026, "Salesforce database: A cloud-native multi-tenant OLTP database," *Proceedings of the VLDB Endowment*, 19(12), pp. 4426–4439.

[5]. H. Song, G. Wei, X. Chen, Y. Feng, X. Fang, Y. Men, H. Feng, Y. Li, and H. Cui, 2026, "K2+: A multi-region OLTP database with disaggregated storage for high availability and strong consistency," *Proceedings of the VLDB Endowment*, 19(12), pp. 3860–3873.

[6]. D. Pereira, F. Pereira, R. Lopes, N. Faria, P. S. Almeida, and J. Pereira, 2026, "Towards abstractions for composable transactional isolation," *Proceedings of the VLDB Endowment*, pp. 1–7.

[7]. H. Ma, and S. Liu, 2026, "A hybrid heuristic–reinforcement learning method for multi-table join optimization in cloud-native databases," *Future Technology*, 5(4), pp. 119–132.

[8]. H. Wu, L. Zhang, X. Zheng, C. Zhu, B. Shang, L. Cao, X. Liu, L. Zhang, and C. Chen, 2026, "Enhancing database write performance with the write operation pushdown framework," *Proceedings of the VLDB Endowment*, 19(12), pp. 3847–3859.

[9]. K. L. Niranjan, V. Dharmar, and K. M. Asraf, 2026, "AI-driven fault-tolerant automated database backup and recovery system," *International Journal of Creative and Open Research in Engineering and Management*, 2(5), Article. 106.

[10]. X. Gao, Z. Liu, J. Cui, H. Li, H. Zhang, K. Wei, and K. Zhao, 2026, "A comprehensive survey on database management system fuzzing: Techniques, taxonomy and evaluation," *ACM Computing Surveys*, 58(10), Article. XX.

[11]. J. Swenson, A. Kimball, R. K. Poss, R. Taft, J. Lim, A. Storm, S. Bhola, P. Bulkley-Logston, P. Tatlow, R. Harding, and R. Shamim, 2025, "CockroachDB serverless: Sub-second scaling from zero with multi-region cluster virtualization," *International Conference on Management of Data*, pp. 648–661.

[12]. B. Lingala, 2025, "Optimizing distributed database performance across geographical boundaries," *International Journal of Research Publications in Engineering, Technology and Management*, 8(5), pp. 12818–12826.

[13]. K. Psarakis, G. Christodoulou, M. Fragkoulis, and A. Katsifodimos, 2025, "Transactional cloud applications go with the (data) flow," *Conference on Innovative Data Systems Research*, pp. 1–7.

[14]. H. Song, X. Chen, R. Gong, Z. Sun, T. Shen, C. Li, H. Feng, S. Wang, and H. Cui, 2025, "Perseus: Achieving strong consistency and high data freshness for scalable geo-distributed HTAP," *ACM on Management of Data*, 3(4), Article. 260.

[15]. A. S. Bhande, N. J. Sharma, and S. V. Barabde, 2025, "Emerging trends in database design: Toward improved modeling and development practices," *EPJ Web of Conferences*, vol. 328, pp. 1–8.

[16]. L. Qiu, R. Taft, A. Shraer, and G. Kollios, 2024, "The price of privacy: A performance study of confidential virtual machines for database systems," *20th International Workshop on Data Management on New Hardware*, pp. 1–8.

[17]. N. K. Miryala, 2024, "Emerging trends and challenges in modern database technologies: A comprehensive analysis," *International Journal of Science and Research*, 13(11), pp. 112–121.

[18]. F. E. R. Rabelo Ferreira, and R. F. do Nascimento Fidalgo, 2024, "A performance analysis of hybrid and columnar cloud databases for efficient schema design in distributed data warehouse as a service," *Data*, 9(8), Article. 99.

[19]. G. Li, W. Tian, J. Zhang, R. Grosman, Z. Liu, and S. Li, 2024, "GaussDB: A cloud-native multi-primary database with compute-memory-storage disaggregation," *Proceedings of the VLDB Endowment*, 17(12), pp. 3786–3798.

[20]. J. Clark, A. F. Donaldson, J. Wickerson, and M. Rigger, 2024, "Validating database system isolation level implementations with version certificate recovery," *Nineteenth European Conference on Computer Systems (EuroSys)*, pp. 754–768.
[21]. tpcc-benchmarks/cockroachdb/README.md at master · mychewcents/tpcc-benchmarks · GitHub