International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 16, No. 1 (Sep-24)

http://dx.doi.org/10.12785/ijcds/1601113

Analysis of Multi-Join Query Optimization Using ACO and Q-Learning

M. P. Karthikeyan[1], K. Krishnaveni[1] and Dac-Nhuong Le[2]

[1]*Department of Computer Science, Sri S. Ramasamy Naidu Memorial College (Affiliated to Madurai Kamaraj University, Madurai), Sattur, Viruthunagr, Tamilnadu, India*
[2]*Faculty of Information Technology, Haiphong University, Haiphong, Vietnam*

Received 8 Feb. 2024, Revised 24 Jun. 2024, Accepted 28 Jun. 2024, Published 26 Sep. 2024

Abstract: The objective of this research is to address the challenges of query optimization in systems with a large search space of execution plans caused by distribution. The research aims to develop effective strategies for optimizing the join operator in a relational database, specifically focusing on minimizing workload delay and query costs. Additionally, the research aims to compare the performance of Ant Colony Optimization (ACO) algorithm and Q-Learning technique in achieving optimal execution plans for queries. To achieve the objective, the research utilizes a reinforcement learning model, specifically the Q-Learning technique. The Ant Colony Optimization (ACO) algorithm is also employed as a comparison method. These techniques are applied to the query optimization process, specifically for adjusting the optimal execution plan for a query in a continually updating environment. The research findings reveal that the utilization of Q-Learning and Ant Colony Optimization techniques improves the query optimization process. These techniques help identify optimistic queries with minimal workload delay and query costs. By applying reinforcement learning and optimization algorithms, the research successfully achieves an optimal execution plan for problematic queries in a distributed system with a large search space of execution plans. The novelty of this research lies in the combination of reinforcement learning and optimization algorithms for query optimization in a distributed system. The utilization of Q-Learning technique and Ant Colony Optimization algorithm in this context addresses the challenges of adjusting optimal execution plans in a continually updating environment. This research contributes to the field by providing effective strategies for optimizing the join operator in a relational database, improving the efficiency of query optimization, and reducing workload delay and query costs.

Keywords: Query optimization, join query, Ant Colony Optimization, reinforcement learning, query execution plan

1. INTRODUCTION

The paper begins with an introduction outlining the challenge of optimizing multi-join queries in distributed systems, introducing Ant Colony Optimization (ACO) and Q-Learning as potential solutions. Section 2 delves into the intricacies of the multi-join query optimization problem, highlighting the complexities involved and the importance of efficient optimization for system performance. Following this, Section 3 elaborates on the application of ACO to multi-join query optimization, detailing its principles and discussions. In parallel, Section 4 introduces Q-Learning as an alternative approach, explaining its workings and comparing it with ACO in terms of efficacy. The experimental results, presented in Section 5, elucidate the outcomes of evaluating ACO and Q-Learning in a controlled setting, focusing on metrics such as optimization time, workload delay, and cost. Finally, the conclusion summarizes the key findings, discusses their implications, and proposes avenues for future research, providing a comprehensive overview of the study's contributions to the field of query optimization in distributed systems.

The method of choosing an efficient execution strategy for handling a query is known as query optimization. A conventional query optimizer calculated the cardinality of the various tables using statistics and probability criteria that were saved, resulting in the finding of the optimum query method. These data also included the dataset, blocks, different values in each column, attribute selectivity, and the typical number of records satisfying an equality condition [1]. Reinforcement Learning (RL), one of the machine learning exact methods, is being used to describe or discuss how such a computational intelligence picks up new information and improves its approach through contact with the environment [2], [3], [4], [5]. Deep Reinforcement Learning is required to determine the best query execution technique. These techniques are advantageous because they can serve as manually programmed query optimizer components and are adaptable to different workloads and data distributions [6]. We investigated these

E-mail address: kkrishnaveni@srnmcollege.ac.in,karthi.karthis@gmail.com, nhuongld@dhhp.edu.vn

DRL-based query optimizers to understand more about their performance limitations. Researchers showed that all such methods have constraints as of their fundamental models consider giving unfortunate estimates and lack a universally applicable feature encoding, despite effectively choosing high quality query plans the rest of the time. we have been unable to connect the open source for past work. According to our supposition, withdrawing these restricts will result in additional maximize the performance for DRL-based information retrieval. RL is an unsupervised method that allows an individual's behavior to observe the state of its surroundings and perform actions that directly effect it in order to develop the best policy. Although Reinforcement Learning can be used to solve optimization problems for a stand-alone treatment, its use to flexibly systematic framework optimization problems in dynamic, decentralized systems are currently being researched.

Many academics have worked hard to enhance the low exploration efficiency of the basic Q-learning algorithm, which is based through valuation, mention strategic plan learning, and the TD approach [7]. The Q-Learning system was enhanced with tutor supervision and a unique relatively modest control-based overseen reinforcement learning path planning reach. This accelerated the system's convergence and established the notion of least in part directed q-learning. It also used the Q-learning strategy to initialize the Q-Table, which improved the route planning process. However, Ant Colony Optimization (ACO), a meta-heuristic optimization tool, employs work load latency in high order performance and is inspired by genuine ant species and their foraging methods. Artificial termites in a colony collaborate to tackle difficult discrete optimization problems. As a result, collaborative effort seems to be an essential piece of something like the artificial ants. It is also acknowledged the said ant colony can just be used in as both a framework such as evolving heuristic methods to handle optimization technique. Ant colony optimization, positive list of basic investigators (artificial ants) is given computational resources and a way to interact with one another [8], [9], [10].

ACO algorithm is an appropriate and efficient option for optimizing a query in databases due to its features and traits such as its resilience, based optimization, parallelism derived from the ability to function simultaneously and independently, and flexibility to combine with other techniques [11]. ACO and RL are distinguished by the fact that ACO employs a heuristic evaluation to identify which moves are superior, whereas RL does not. ACO also employs a distributed method, in which numerous agents collaborate to solve a specific problem. The ACO algorithm's low order performance is work load delay, and this may last distinction—the ACO's use of heuristics—made completely modeling an ACO algorithm as an RL algorithm challenging.

A. *Query Optimization*

The classic difficulty of multi-query optimization is RDF/SPARQL. For the problems experienced during optimization, we can state that, first and foremost, if an optimizer is unable to continue exploring the search space, they cannot be certain of discovering the ideal execution strategy. As a result, we must devise a method for searching the entire search space. Second, many traditional models struggle to generate reliable cost estimates, and joining ordering— which governs the rate of data flow— is a critical issue. So, the order of joins has a considerable impact on query performance [12]. We analysis an optimization solution for the join ordering problem in this study in order to address the aforementioned problems; more details are provided in the section following.

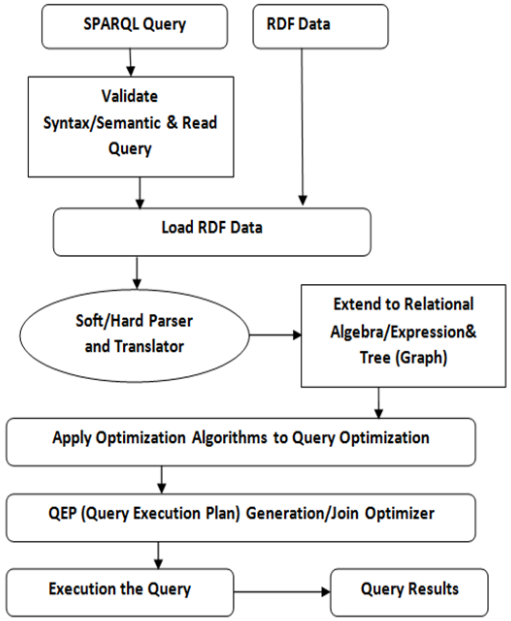

Figure 1. Complete process of query optimization

Figure 1 depicts the full query optimization technique. The procedure starts with an input SQL query, which is then passed to the parser, which generates an algebra tree linked to the query. Our algorithms then select the optimal join order from among numerous join order options. The join order is used to generate a query execution method (QEP). The query execution plan is traversed by the execution engine, and the appropriate retrieved data is output.

To process a query, a sequence of steps must be taken: first, open up the request to really be filtered; then, optimize the request's inside of. finally, run the optimized request through the engine to obtain the consequence [13], [14]. Every aspect of the query's processing is demonstrated step by step here.

Create the query first, then send it to the query engine for processing. Take a look at a SPARQL, SQL, or RDF query.

The semantic test can be used to assess a statement's meaning. If the shared pool contains code, the shared pool check recognizes it and does not perform any additional database optimization or execution steps.

The data retrieval process and after that debugs it and application into to the format provided regarding regular expression parsing comparison or keyword order, such as `SELECT`, `WHERE`, and so on, and verifies for the RDF attribute. Side by side comparisons of RDF repositories These RDF repositories are query able.

The analyzer then modifies the triple pattern order of the original question to match that of its equivalent form in SPARQL algebra.

Check the linkages and syntax. Provide an equivalent relational algebraic expression for the question.

Query can receive results predicated on the RDF repositories' primary model without any optimization, but as the repositories increase in size, it becomes required to execute the query using various optimization techniques to ensure that it produces results in a reasonable amount of time. Using the type of approach, the simplest level of work is chosen from among well all query plans.

Query optimization is a technique in which a database system assesses numerous query approaches and chooses the best optimal strategy with the lowest expected cost. Throughout the optimization process, each statement should go through hard parsing at least once, and the parsing should be enhanced. A repeating execution plan is generated by Row Source Generator using the optimizer's execution plan software and the rest of the database. A set of rows are produced by each iterative execution phase, and the application that issued the SQL statement receives rows in the final step.

Develop a low-cost evaluation method for the question plan.

If there have been no errors up to this point in the process, the final step is the query execution phase, in which an optimized query is run through a query engine to receive the results of a SPARQL query.

2. Multi-Join Query Optimization Problem

The following is an explanation of how an RDBMS handles user queries: After receiving the query, the query parser analysis its grammar, checks relationships, and turns the user-submitted request into its internal form. it is usually turned into a relational formula expression, often known as a query syntax tree. a structured algebra affirmation has many equivalent expressions, and thus many equivalent query linguistic structure trees. Following that, the query optimizer determines which physical implementation to utilize for each relational algebra operation and generates the query execution plan (QEP). The QEP describes the physical approach that will be used for each operation, as well as the order in which the operations in A query will be performed. The query-execution engine takes and executes the QEP that has been passed to it, and returns the results to the user. The query optimizer chooses the one with the lowest cost output to the engine from among all similar QEPs. Once the optimization step of applying pressure the to choose and project operations is implemented, a query that comprises project, select, and join operations would then transform into a structured algebra expression made up of n join operations, referred to as a join tree [15], [16].

An example numerous query q encompass $r_1; r_2; r_3; r_4; r_5$ five relations, that joined as Figure 2 left deep trees, Figure 3 shown Bushy tree and Figure 4 shown right deep tree. These are signified as some using of join plant. The nodes are relations comprising query q and the intrinsic nodes express join operation and significant improvement

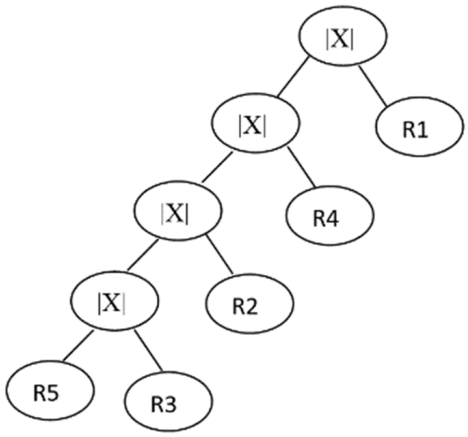

Figure 2. Left-Deep tree (Join Tree Q1)

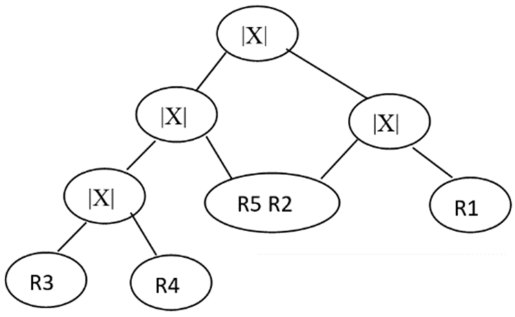

Figure 3. Bushy tree

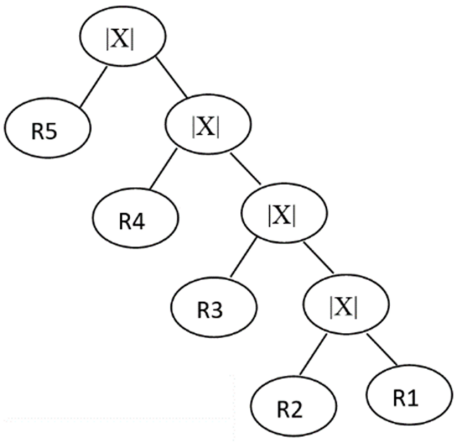

Figure 4. Right Deep Tree (Join Tree Q2)

Bottom-up execution refers to executive orders. The physical methods used to perform the join operation in a different order and with a different physical result in considerable variations in the cost of the connect trees. Assuming that all join operations are performed using the same physical technique, the problem of optimizing multi-join queries can be reduced to choosing the best join order and resulting join tree from a cost standpoint. Given that trees in left linear space can completely utilize this same measurement and frequently contain the best strategy, or at smallest a method with a cost equal to the optimal strategy, consider left relevant as a random search.

Attempting to prevent the mathematical product's beginnings is commonly regarded as a difficulty constraint in order to narrow the search area even further. Q is an example of a multiple join query with five relations: $R_1; R_2; R_3; R_4; R_5$.

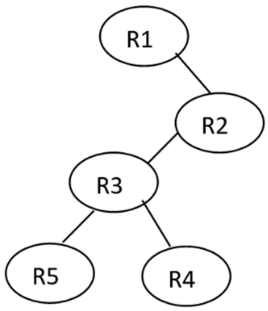

Figure 5. Query graph

Figure 5's query graph $G = (V, E)$ illustrates the qualities associated with five relations based on statistical data from the database catalogue. The query graph's Relationships are illustrated as nodes, and an edge between two denotes the characteristics they share. Figures 2 and 4 depict the two join trees Q_1 and Q_2 in the query Q's left linear space. When the "avoiding Cartesian product" restriction constraints are applied, Q_1 fails to meet the conditions and is an invalid join tree; Q_2 satisfies the conditions and is a suitable joining tree.

The query optimizer must choose appropriate Order of Execution of Relational Operators, Access Methods for Relevant Relations, and Interactions for Join Operations again from specified solution space in order to improve the set of metrics of the resulting Request Execution Plan by implementing the appropriate search strategy. Establish the order of transmitting data across locations to minimize data volume and network connectivity costs [17]. Information retrieval methods have been the focus of research in distributed databases. because of their ability to search globally and their successful application to diverse combinatorial optimization problems, a new class of methods, including iterative methods, ant colony optimization, particle swarm optimization, and many more, are currently being utilized in structured and distributed databases to identify optimal and suboptimal solutions for big join queries in the provided problem space. Query Optimization makes assumptions and unifies cost models by portraying the price of a query plan as a recurring process, which is how we introduce optimization algorithms using query optimization in this paper and define and analyze corresponding query optimization methods. The next examples require this since it allows for the modeling of numerous parameters and cost measurements.

A. *Scenario 1*

As a result of a cloud service, multiple user can request a large scientific data gathering via a interface. requests are processed in the cloud. multiple user publishes user requests by joining predicates onto the web interface, that conform to query templates such as

```
SELECT * FROM Table-Name WHERE Q1 AND Q2
```

where Q_1 and Q_2 are unnamed predicates. Accepting higher monetary fees can frequently shorten the time it takes to process requests in the cloud [18].

As both a result, after submitting a query, customers are presented with a visual representation of various exchange among completion time and financial costs (as realized by other query plans), from which they can select the appropriate exchange. to hurry up the production, its cloud service undertakes a preprocessing step in which it determines the applicable request intends by each request framework. both expense performance measures were indeed processing time or pecuniary costs since the selectivity's of the predicates must be specified as parameters during preprocessing. It is deemed relevant if the time-fees tradeoff of a request intend was indeed pare to-optimal at for lesser it only variable inside this measurement space—that seems to be, no exit option must have fees as well as way quicker runtime.

B. Scenario 2

Integrated SQL queries are a typical application for P[Q2; Q3]: All pertinent request strategies for one particular request format were also approximated ahead of schedule versus eliminate information retrieval pricing sometimes when speed. Parameters model the quantity of run-time bu er space or the selectivity of irrelevant predicates. the processing time is the only cost metric in the typical situation. however, inside the case of interpolation method query processing, execution time may well be exchanged for result precision [15]. therefore, in scenario, optimization must take both the metrics execution time and result precision into consideration. the optimum request plan is decided at runtime based not just on actual values are provided and more on a regulation and it calculates how to best balance execution time and return precision, for example, depending on the present total power or the lowest accuracy needed for such a single iteration.

Algorithm 1 Ant Colony Query Optimization

Input: The query relations $s = fr_i g(i = 1::n)$
Output: The optimization of query relations.
BEGIN
1) Step 1: Con gure the speci cations. The suggested algorithm's parameters are set to zero.
2) Step 2: Sorting the nodes by their geographical coordinates: These nodes are classi ed as either perimeter nodes or center nodes. In preparation for the algorithm's subsequent execution, the categorized results are recorded and sorted.
3) Step 3: Then m ants are assigned at random to one of the n nodes, and this node is added to the ant's tabu list.
4) Step 4: The path length is calculated when the ants have made their choice. after that, the relevant tabu list is altered. rep Step 3 until about the ant has accomplished its tour. the current optimal path length is retained in this iteration, and the global optimal path is updated.
5) Step 5: Refresh the secretion: using the equation from the upgraded secretion updating rules, the pheromone on the optimal path is worldwide modi ed.
END.

3. Ant Colony Optimization

The ant colony is really a meta-heuristic algorithm called secretion data in order to identify a quickest distance from with a protein source here to hives without use of impactful cues. Ant colonies leave behind pheromones, which are chemical molecules, as they seek for food. More pheromone is released onto the ground as the ants go down the track. The above positive reaction system is likely to result in the formation of an obvious direction because after with their hives towards the source of food, enabling the next ant to choose one path with a probability related to the amount of pheromone. The following are the primary traits of actual ants:

(1) True ants choose the strongest pheromone trail.

(2) The amount of pheromone left on a path reduces the distance.

(3) Pheromones allow genuine ants to communicate in an indirect way.

The ACO was created using the previously indicated behavior of identifying genuine ants looking for the shortest path. Arti cial ants collaborate to solve problems by exchanging information through the use of pheromone deposits on paths. This revolutionary method is known as Ant Colony Optimization (ACO). Numerous arti cial ants help ACO build optimization methods and exchange information via a communication system. They take the shortest paths because ants emit pheromones that other ants can detect and follow if they come across them. As a result, the collective behavior that has established shows that if a few ants choose the same course of action, the likelihood of further ants choosing the same course of action increases. The key idea behind ACO is to portray a problem as a graph search for the shortest path with the lowest cost [18], [19].

The following are the steps in the Ant Colony Query optimization algorithms:

Iteration control is the sixth step. set the iterative counter to return to Step 4. if the procedure is not rescinded, the best workaround is returned [20], [21].

Limitations of ACO Ant colony does have strict limitations whereas getting special contains somewhere around parallelization, computing, global network view abilities, but instead fast speed, to name few of [22], [23]:

> The ACO-required 1st structure appears to lack the one organized format sure going to begin. along secret mission sure nutrition, that whole of such drive over all locations once spontaneous, leading to the one severely restricted community scattering potential. such an elongates a timing provide the optimum solution.
>
> Because of the low pheromone concentration on the ants' traversal pathways, the initial convergence speed of ACO is reduced. this lengthens the time required to produce a set of interesting possibilities. however, because of the positive feedback mechanism, the pace of convergence to the optimized design rises.

The ACO algorithm iteration is frequently separated into two sections. To begin, both ant in the colony leaves a path in the graph from a starting vertex to the end vertices and edges. Second, once all ants have begun to arrive at the completion vertices and edges, the perimeters of each path are marked with a quantity of secretion proportional to the standard of the path reported. In the conventional ant system [24], an ant k establishes a path by recursively commuting from its most recently visited vertex x to some other vertices and edges y that the ant has yet to witness. Estimated probability $p^k_{xy}(i)$ for an ant k to migrate from vertex x to an unvisited vertices and edges y at iteration I of the process is de ned as [25]:

$$p_{xy}^{k}(i) = \begin{cases} \dfrac{[T_{xy}(i-1)]}{\sum_{z \in V_k(x)} [T_{xz}(i-1)]}, & z \in V_k(x) \\ 0, & z < V_k(x) \end{cases} \quad (1)$$

As between, the and parameters determine the relative importance of the information given by global secretion traces and local heuristics. The secretion amount $T_{xy}(i)$ which is coupled with an edge connecting vertices x and y, is updated at iteration i based on the most recent experience of all ant colonies. Where $T_{xy}(i-1)$ is the general secretion amount on the edge between vertices x and y at iteration process $T_{xy}(i-1)$ is a neighborhood methodology measure capturing the inverse of the (estimated) distance between vertex x and vertex y, and $v_k(x)$ is the ant k's unvisited vertices after visiting vertex x.

$$T_{xy}(i) = (1-)T_{xy}(i-1) + \sum_{k=1}^{m} T_{xy}^{k}(i) \quad (2)$$

where denotes an extraction e iency that prevents the colony from correlating to a local optimum and $T_{xy}^{k}(i)$ records the amount of secretion deposited by ant on an edge $!_{xy}$ connecting vertices x and y at iteration i. At repetition I this quantity is provided as a function of a constant q and the length L_k of ant k's path $T_k(i)$. i.e.,

$$T_{xy}^{k}(i) = \begin{cases} \frac{Q}{L_k}, & !_{xy} \in T_k(i) \\ 0, & !_{xy} < T_k(i) \end{cases} \quad (3)$$

4. Q-Learning

A crucial RL concept is Temporal-Di erence (TD) learning. It combines techniques from Monte Carlo (MC) and Dynamic Programming (DP). TD, like MC, learns directly from raw experience and modi es estimates without waiting for the nal result [24]. One of these TD control algorithms which discovers the action-value function Q stands for Q-learning $s(a)$. This function tells the agent whether doing action a while in states is a good idea. To isolate the acting policy from the learning policy, Q-learning adopts an o-policy method. As a result, even if the action chosen in the following state was poor, the information was not incorporated in the update of the current state's Q-Function, resulting in a terrible decision [26]. However, because Q-learning employs -policy, the di culty is resolved.

The Q-value calculation is as follows:

$$Q(x;a) = R(x;x^{0};a) + \sum_{x^{0}} P(x^{0}|x;a) \max_{a2A} Q(x^{0};a) \quad (4)$$

The value represents the learning rate, which ranges from 0 to 1. r is indeed a prize a certain denotes the speed with which it is curtailed accented with oil - rubbed period. $Q(x;a)$ of intervention with in present incarnation a amount of the present virtues $\frac{1}{4}(s;a)$ and or the equations clearly indicates the simplest activity inside the current condition are also used to keep updating Q-learning has been preserved besides using such this same past equation of between frequently upgrade the q-value for every situation. prior to commencing Q-Learning, its q-table includes trophies. whether an operator inside this condition determines doing an activity oriented on something like a strategy, that as well advancements towards the next government (4). the above step is conducted loads of times only until outcome q-value iterates to just a unique value, about which direction that whole q-table is being used to remedy of one problem situation. To solve the Bellman problem, Q-Learning integrates whether nonlinear optimization but also model - based methodology. Q-learning seems to be the core element among several classi cation methods as its simplicity or fruitful such as single person particular conditions. a value would be only recon gured once each activity out q learning. subsequently, — when it comes myriad states-actions would've never been witnessed already when, this is hard complete aim to address complex di culties productively in such a expansive state-action setting. besides this, because of the table such as rewards would be predetermined, a large quantity after all memory rack is required. together in inter system consists of two and more agencies, an outsized state-action brain seems to be obligated, where it postures troubles. As a nal outcome, foundational q-learning methods have been made obsolete so they are powerless sure achieving successful learning together in multi-agent particular circumstance [27].

Q-Learning algorithm : As previously stated, q-learning is among the most thoroughly was using learning algorithms [28]. That whole method is easy and also is primarily based on continuing to follow updating formula:

Algorithm 2 Q-Learning Algorithm

BEGIN
1) Initialize function of Q-arbitary
2) Take note of the present situation s.
3) Times
   Using Q, compute the policy $s(a)$.
   Select an action an in accordance with agent policy $s(a)$.
   Carry out action a and note the resultant reward and the following
   States.
   Apply Equation (4) to update $Q(s;a)$.
   Set $s ! s$.
4) Till completed.
END.

Agent Training process The agent must explore his surroundings and learn from his behaviors, whether positive or negative, when using an algorithm like Q-learning. The key principle of this strategy is to compute the maximum each state's predicted future bene ts for ot [23]. The

Q table, which is an important idea in this strategy, then appears. In fact, this table retains the Q-value obtained from each action-state and hence depicts the environment to some extent. This value is calculated by the Bellman equation [29] from the expected cumulative reward, where represents the discount rate ( ), i.e. the greater it is, the more important distant rewards are

$$Q(s,a) = E\left[\sum_{t=0}^{s} R_{t+1}(s,a)\right] \quad (5)$$

Q-values for a given state and action, the value for projected discounted cumulative reward $R_{t+1}(s,a)$. In the first part of the operation, the Q-table must be built with the following parameters: n columns for the number of actions and m rows for the number of states, with total values set to zero. Then comes the decision and action. The concept of exploration and exploitation trade-off enters the picture with the "-greedy strategy." As earlier mentioned, the "parameter will be higher at initially and will decrease progressively." The agent discovers the environment for the first time, according to the reasoning. And the second time, he'll apply his expertise to make the best decisions possible. The final phase is the update, which comes after the action has been chosen and executed.

Q-Table: The Q-table must be changed with the value of the action-state, where is the learning rate ( ) which averages modifications.

Figure 6. Example of a basic environment (left) and its Q-table(right)

Measure Reward and Evaluate Now we have taken an action and observed an outcome and reward.

$$Q_{new}(s,a) = Q(s,a) + \quad R(s,a) + \max_{a'} Q(s',a') \quad Q(s,a) \quad (6)$$

The state and action's Q-value $R(s,a)$ - the remuneration for doing an activity in a state; $\max Q(s',a')$ denotes the greatest estimated future payo $Q(s,a)$ denotes the current Q values.

This learning approach can be performed as many times as necessary, until the agent has enough information to always choose the correct action from the Q-table. Figure 6 illustrates a world with only nine states. Consider a four-elevator, ten- oor elevator group control [30], [31]: it generates a lot of state and a large Q-table. Because of the processing power, using the Q-learning method to these cases is impossible. As a result, Deep Learning and Reinforcement Learning have been integrated.

5. Experimental Results

This section will demonstrate numerous successful tests to assess the efficacy of our suggested paradigm. In addition, the latency In any case, our simulations will be compared against the usual inference style of play to determine effectiveness. Mohsin et al. [32] employs a new rendition such as C# interface to perform type evolution technique (ACO) automated system with Q-Learning methodology published via Windows Phone code mag [33], [34], [35], [36], [37]. The software is written in C# and runs on MATLAB. To properly implement the platform version on Hadoop 2.4.1, and Hive 0.12.0, Apache, Spark 1.1.0, as well as to investigate the many factors that may influence the overall performance of encrypted query database systems (see Table I).

Some of the criteria that perform include optimization time, workload delay, minimal cost, scaling up queries size and number of entries solely within the database, request result size, or use of multiple-condition searches, etc. The studies' default values and parameters are bandwidth - 1000 mbps, latency - 10 milliseconds, database size - 50000 records, query size - 1000 records, record size - 10 bytes, and data distribution - same performance. The database system will be assessed based on query efficiency, which will be displayed with optimization time delivery, work load delay, and lowest cost applied.

Optimization Time: The optimization times varies between the ACO and Q-Learning approaches. Keep in mind that the cost estimates are normalized to the best plan discovered. This includes the optimal answer in almost all instances up to size 40, i.e., the costs are correct, but due to the NP-hardness, we can no longer guarantee optimality for larger questions. ACO algorithms appear to improve quicker than Q-Learning methods: sometimes when length 100, it is simply the best method available, whereas Q-Learning approaches fail. Nevertheless, as demonstrated with the smaller sizes, it works well enough in overall, even when regular Q-Learning would have been a superior alternative. the workload optimization time is seen in Figure 7.

The relative advantages of the optimization techniques remain unchanged from earlier workloads. Except for the fact that the ACO Method now produces slightly better plans than the Q-Learning algorithm, the optimization time for the ACO Algorithm is practically linear. Considering the presence of a little super-linear component.

TABLE I. Experimental Parameters

Parameters	Meaning
Software	C# and runs on MATLAB
Platform	Hadoop 2.4.1, and Hive 0.12.0, Apache, Spark 1.1.0
Bandwidth	1000 mbps
Latency	10 milliseconds
Database size	50000 records
Query size	1000 records
Record size	10 bytes

Figure 7. Optimization time

Cost Model. Minimum Cost As a result, because the calculation approach is based on either duration (the total amount of all element values), the same computation has been employed. The general fee is decided even as a total amount anyway /o charge with all participating system and data transport (communication) charge if attempting to transfer organization's between both locations appears to be required:

$$TotalCost = IO_{join} + COM_{Ri} \quad (7)$$

in which,

IO_{join} represents the cost of the join procedure, and

COM_{Ri} indicates the cost of moving entity R_i between site locations.

The IO_{join} cost is calculated as follows:

$$IO_{join} = P_{join} + P_{write} \quad IO(s_c) \quad (8)$$

in which

$IO(s_k)$ signi es a i/o period for such turntable sometimes when postures s_c,

P_{write} signi es that whole pro le rely essential to save all the enlist nal outcome, as well as R_{join} connotes its document number activated of about operate it and enter protocol for both r_i or typically begin.

P_{join} has been derived just like:

$$P_{join} = P_{Ri} \quad P_{Rj} \quad (9)$$

where

P_{Ri} is the number of pages in entities R_i and R_j.

P_{write} is calculated as follows:

$$P_{write} = \frac{card(R_i \text{ join } R_j) \quad len(R_i \text{ join } R_j)}{Ps} \quad (10)$$

where P_s is the size of the page, The average duration of a tuple in R_i is given by $len(R_i)$, while the number of tuples in R_i is given by $card(R_i)$. Overall cost of moving relation R_i from place S_k to location S_p is computed as follows:

$$COM_{R_i} = card(R_i) \quad len(R_i) \quad COM\ S_k; S_p \quad (11)$$

where COM_{R_i} is the time needed to move one byte from location S_k to location S_p.

The ACO model's least values were compared to the Q-learning algorithm's cheapest cost in the rst experiment, as illustrated in Figures 8 and 9. As illustrated by the gures, ACO still provides a lower cost than Q learning. The results show that ACO achieves greater performance in terms of cost minimization.

Work load Latency: Work ow performance, tail latency, and total costs while using the ACO, Q-Learning algorithm to select the query plan with the lowest cost value. hinting that ACOs are better than Q-Learning methods for changing hardware Figures 9 depicts a comparable comparison to a commercial database system. Again, ACO can save money and reduce workload delays. The ordinance, however, is small, and the overall costs are much lower, signaling that the commercial system is a better starting point than the Q-Learning method. When compared to the Q-Learning algorithm, for example, ACO achieved a

cost and time reduction of over 50%, meaning that the commercial system is more exible to various hardware.

Figure 8. Minimum cost

These expenses do not include commercial system li censing fees, which were waived for the purposes of this study. Work load until a set of queries is done, latency performance has a signi cant impact on dashboard perfor mance), we will look at the distribution of query latencies w Figure 9 compares the workload latency of Q-Learning and ACO algorithms, which is the amount of time required to build a query plan. Despite the fact that ACO avoids the Q-Learning algorithm's accelerating lookup expense, and as such it is somewhat easier than Q-Learning for requests due to the price of creating network calls. Overhead is outweighed by the exponential search cost as the number of relations to join grows, and the ACO optimization delay climbs substantially quicker.

Figure 9. Workload latency

Scale up Queries throughout this number of experiments, researchers strong impact after all person improvement like grow exponentially asks. In our testing, we look at the ACO algorithm rst, then the Q-Learning algorithm. Each time an ACO was recalculated, only about 45 bene ts were recalculated. In comparison, the Q-Learning algorithm does around 1558 bene t re-computations per time, resulting in a query optimization time of seventy-seven seconds, as opposed to seven seconds for ACO. Scale up is another major concern in the Q-Learning process. The ACO plans also produced roughly identical results on the queries we ran.

As a result, the ACO saves substantial time while maintaining the quality of the designs presented. Researchers calculated the price of ACO without the sharability data processing to determine the advantage of the sharability computation; each node is assumed to be potentially cacheable. We discovered that the optimization time grew dramatically across the spectrum of scale up queries, and also that sharability computation is also a highly helpful improvement. Because the optimization time for the Q-Learning method and the ACO algorithm is linear, the ACO algorithm now produces slightly better plans than the Q-Learning algorithm. The minimum cost ACO model was compared to the minimum costs achieved using the Q-Learning method. In summary, our modi cations of the ACO's implementation result in an orders of magnitude gain in performance, which is critical for its utility.

6. Conclusions and Future Work

Query optimization has a signi cant impact on the cost. In this study, we provided an optimization of reinforcement learning execution using the Ant Colony method and the Q-Learning algorithm. The lowest cost, workload delay, optimization time, and scale up questions, according to study. According to the experimental data, ACO outperforms Q-Learning in terms of discovering optimal solutions in terms of both time and quantity. The in uence of query cost and optimization time will also be investigated. When compared to the Q-Learning algorithm, using a non-dominated ACO algorithm can discover optimistic queries and reduce query cost.

Future work could focus on re ning and extending the reinforcement learning models used in this study. Exploring advanced algorithms or hybrid approaches that combine multiple reinforcement learning techniques could lead to further improvements in query optimization performance. As new technologies and frameworks for distributed computing continue to evolve, future work could explore the integration of reinforcement learning-based query optimization techniques with emerging platforms such as edge computing, server less architectures, or block chain-based databases.

References

[1] G. Antoshenkov, "Dynamic query optimization in rdb/vms," in Proceedings of IEEE 9th International Conference on Data Engineering. IEEE, 1993, pp. 538–547.

[2] X. Zhou, T. Bai, Y. Gao, and Y. Han, "Vision-based robot navigation through combining unsupervised learning and hierarchical reinforcement learning," Sensors, vol. 19, no. 7, p. 1576, 2019.

[3] S. Krishnan, Z. Yang, K. Goldberg, J. Hellerstein, and I. Stoica, "Learning to optimize join queries with deep reinforcement learning," arXiv preprint arXiv:1808.03196, 2018.

[4] R. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil, and N. Tatbul, "Neo: A learned query optimizer," arXiv preprint arXiv:1904.03711, 2019.

[5] R. Marcus and O. Papaemmanouil, "Deep reinforcement learning for join order enumeration," in Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, 2018, pp. 1–4.

[6] ——, "Towards a hands-free query optimizer through deep learning," arXiv preprint arXiv:1809.10212, 2018.

[7] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.

[8] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," IEEE computational intelligence magazine, vol. 1, no. 4, pp. 28–39, 2006.

[9] G. C. Onwubolu and B. Babu, New optimization techniques in engineering. Springer, 2013, vol. 141.

[10] D.-N. Le, "A new ant algorithm for optimal service selection with end-to-end qos constraints." JIPS, vol. 18, no. 5, pp. 1017–1030, 2017.

[11] T. Dökeroğlu and A. Coşar, "Dynamic programming with ant colony optimization metaheuristic for optimization of distributed database queries," in Computer and Information Sciences II: 26th International Symposium on Computer and Information Sciences. Springer, 2012, pp. 107–113.

[12] W. Le, A. Kementsietsidis, S. Duan, and F. Li, "Scalable multi-query optimization for sparql," in 2012 IEEE 28th International Conference on Data Engineering. IEEE, 2012, pp. 666–677.

[13] E. G. Kalayci, T. E. Kalayci, and D. Birant, "An ant colony optimisation approach for optimising sparql queries by reordering triple patterns," Information Systems, vol. 50, pp. 51–68, 2015.

[14] M. Meimaris and G. Papastefanatos, "Distance-based triple reordering for sparql query optimization," in 2017 IEEE 33rd International Conference on Data Engineering (ICDE). IEEE, 2017, pp. 1559–1562.

[15] A. N. Swami and B. R. Iyer, "A polynomial time algorithm for optimizing join queries," in Proceedings of IEEE 9th International Conference on Data Engineering. IEEE, 1993, pp. 345–354.

[16] F. Q. Cao, Y., "Parallel query optimization techniques for multi-join expressions based on genetic algorithm," Journal of Software, vol. 13, no. 2, pp. 250–257.

[17] H. Yinghua, M. Yanchun, and Z. Dongfang, "The multi-join query optimization for smart grid data," in 2015 8th International Conference on Intelligent Computation Technology and Automation (ICICTA). IEEE, 2015, pp. 1004–1007.

[18] H. Kllapi, E. Sitaridi, M. M. Tsangaris, and Y. Ioannidis, "Schedule optimization for data processing flows on the cloud," in Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, 2011, pp. 289–300.

[19] A. Hulgeri and S. Sudarshan, "Parametric query optimization for linear and piecewise linear cost functions," in VLDB'02: Proceedings of the 28th International Conference on Very Large Databases. Elsevier, 2002, pp. 167–178.

[20] P. Bizarro, N. Bruno, and D. J. DeWitt, "Progressive parametric query optimization," IEEE Transactions on Knowledge and Data Engineering, vol. 21, no. 4, pp. 582–594, 2008.

[21] S. Agarwal, A. P. Iyer, A. Panda, S. Madden, B. Mozafari, and I. Stoica, "Blink and it's done: interactive queries on very large data," 2012.

[22] Z. Zukhri and I. V. Paputungan, "A hybrid optimization algorithm based on genetic algorithm and ant colony optimization," International Journal of Artificial Intelligence & Applications, vol. 4, no. 5, p. 63, 2013.

[23] M. Nasiraghdam, S. Lotfi, and R. Rashidy, "Query optimization in distributed database using hybrid evolutionary algorithm," 2011 International Conference on Information Retrieval & Knowledge Management (CAMP). IEEE, 2010, pp. 125–130.

[24] M. Gregory, "Genetic algorithm optimisation of distributed database queries," in 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360). IEEE, 1998, pp. 271–276.

[25] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," IEEE transactions on systems, man, and cybernetics, part b (cybernetics), vol. 26, no. 1, pp. 29–41, 1996.

[26] P. Dayan and C. Watkins, "Q-learning," Machine learning, vol. 8, no. 3, pp. 279–292, 1992.

[27] L. Shoufeng, L. Ximin, and D. Shiqiang, "Q-learning for adaptive traffic signal control based on delay minimization strategy," 2008 IEEE International Conference on Networking, Sensing and Control. IEEE, 2008, pp. 687–691.

[28] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," Robotica, vol. 17, no. 2, pp. 229–235, 1999.

[29] R. Bellman, "The theory of dynamic programming," Bulletin of the American Mathematical Society, vol. 60, no. 6, pp. 503–515, 1954.

[30] H. Crytes Robort and G. Barto Andrew, "Elevator group control using multiple reinforcement learning agents," Boston: Kulwer Academic, 2002.

[31] F. Lu, P. G. Mehta, S. P. Meyn, and G. Neu, "Convex q-learning," in 2021 American Control Conference (ACC). IEEE, 2021, pp. 4749–4756.

[32] S. A. Mohsin, S. M. Darwish, and A. Younes, "Dynamic cost ant colony algorithm for optimize distributed database query," In-