SDD-1 Algorithm

Bachelor of Technology
Computer Science and Engineering

Submitted By

ARKAPRATIM GHOSH (13000121058)

February 2024

Techno Main Salt Lake
EM-4/1, Sector-V, Kolkata- 700091
West Bengal
India

TABLE OF CONTENTS

1. Introduction

Distributed query optimization is a crucial aspect of managing databases in distributed computing environments where data is spread across multiple nodes or locations. It involves the optimization of queries that access data distributed across different sites or databases to minimize response time, resource utilization, and network traffic. The goal is to improve the overall performance and efficiency of distributed database systems.

Challenges in Distributed Query Optimization:

- Data Distribution: Data in a distributed database may be fragmented or replicated across multiple sites, leading to increased complexity in query processing and optimization.
- Heterogeneity: Distributed databases often involve heterogeneous systems, such as different hardware platforms, operating systems, and database management systems (DBMS), which can complicate query optimization.
- Communication Costs: Communication between distributed nodes incurs overhead in terms of latency, bandwidth usage, and network congestion. Optimizing queries to minimize data transfer over the network is essential for improving performance.
- Data Localization: Query optimization must consider the location of data and minimize the movement of data across distributed sites to avoid unnecessary data transfer and improve query response time.

Strategies for Distributed Query Optimization:

- Fragmentation and Replication: Distributing data across multiple sites through fragmentation (partitioning) and replication (copying) can improve query performance by reducing data transfer and improving data locality.
- Cost-Based Optimization: Cost-based optimization techniques analyze query execution plans based on factors such as data distribution, network latency, and processing costs to select the most efficient query execution strategy.
- Parallel Processing: Distributing query processing across multiple nodes and executing query fragments in parallel can improve performance by leveraging the processing power of distributed resources.
- Query Decomposition and Distribution: Decomposing complex queries into smaller sub-queries and distributing them to relevant sites for execution can reduce response time and network traffic.
- Data Caching and Materialized Views: Caching frequently accessed data and pre-computing query results using materialized views can improve query performance and reduce the need for expensive distributed query processing.

- Global Query Optimization: Optimizing queries across multiple sites simultaneously by considering global query plans and distributed query execution strategies can further improve performance in distributed environments.

Example of Distributed Query Optimization:

Consider a distributed e-commerce platform with customer data stored in a central database and product data distributed across regional databases. When a customer searches for a product, the query needs to access both customer and product data distributed across different sites. Distributed query optimization techniques can be applied to minimize data transfer and query response time by:

- Fragmenting product data based on region and replicating frequently accessed data.
- Decomposing the query into subqueries targeting relevant product databases.
- Leveraging parallel processing to execute sub-queries in parallel across distributed nodes.
- Using cost-based optimization to select the most efficient query execution plan based on factors such as network latency and data distribution.

By applying these strategies, the distributed query optimization process can significantly improve the performance and efficiency of query processing in distributed database systems, enabling scalable and responsive data access across distributed environments.

2. Body

2.1. Semijoin Based Approach

The SemiJoin-based approach is a technique used in distributed query optimization to minimize data transfer and improve query performance in distributed database systems. It involves reducing the size of intermediate data exchanged between distributed sites by performing partial joins before transmitting data across the network. This approach is particularly useful when executing join operations involving large datasets distributed across multiple nodes.

Key Concepts in SemiJoin-based Approach:

- SemiJoin Operation: A SemiJoin operation computes the join between two relations but only returns the attributes of one relation, based on the matching tuples in the other relation. It is denoted as $R \ltimes S$, where R and S are two relations, and $\ltimes$ represents the SemiJoin operator.
- Join Semantics: The SemiJoin operation preserves only the tuples from the first relation (R) that have matching tuples in the second relation (S). It eliminates non-matching tuples from R before transmitting data across the network, reducing the volume of data transferred.

- Optimization Goal: The goal of the SemiJoin-based approach is to minimize the amount of data transmitted over the network during distributed query processing, thereby reducing communication costs and improving query performance.

Steps in SemiJoin-based Approach:

- Query Decomposition: The original query is decomposed into sub-queries that can be executed independently at different distributed sites.
- SemiJoin Transformation: SemiJoin operations are applied to join operations in the query plan. Instead of performing the full join operation, only the attributes of one relation are preserved based on the matching tuples in the other relation.
- Local Processing: Each distributed site processes its portion of the query independently, performing SemiJoin operations to filter and reduce the size of intermediate data before transmitting it to other sites.
- Data Exchange: Intermediate results from SemiJoin operations are exchanged between distributed sites over the network. Since the size of data has been reduced through SemiJoin operations, the volume of data transferred is minimized.
- Final Join Operation: Once the intermediate results have been exchanged, the final join operation is performed using the reduced datasets to produce the final result of the query.

Example of SemiJoin-based Approach:

Consider a distributed database system with two sites: Site A and Site B. Each site stores a large table of orders (Order_A and Order_B) and a smaller table of customers (Customer_A and Customer_B). To perform a join operation to find orders placed by customers at both sites, the SemiJoin-based approach can be applied as follows:

- Query Decomposition: Decompose the join query into sub-queries, with each site processing its portion of the data independently.
- SemiJoin Transformation: Apply SemiJoin operations to filter the orders table based on matching customer IDs. For example, at Site A, perform Order_A ⋉ Customer_B to filter orders by customers at Site B.
- Local Processing: Each site processes its SemiJoin operation locally, reducing the size of intermediate data by eliminating non-matching tuples.
- Data Exchange: Exchange the reduced intermediate data between sites over the network. Since the data size has been reduced through SemiJoin operations, the volume of data transmitted is minimized.
- Final Join Operation: Perform the final join operation using the reduced datasets to produce the final result of the query, which contains only the orders placed by customers at both sites.

By leveraging the SemiJoin-based approach, the distributed query optimization process minimizes data transfer and communication costs, resulting in improved query performance and efficiency in distributed database systems.

2.2. Hill Climbing Algorithm

The Hill Climbing Algorithm is a popular optimization technique used in artificial intelligence and machine learning for solving optimization problems. It is a local search algorithm that iteratively improves a solution by making incremental adjustments to it, aiming to reach an optimal or near-optimal solution. The algorithm is inspired by the metaphor of climbing a hill, where one tries to ascend to the highest point by continuously moving in the direction of the steepest slope.

Basic Principle:

The basic principle of the Hill Climbing Algorithm is to start with an initial solution and iteratively make small modifications to it to improve its quality. At each iteration, the algorithm evaluates the current solution and selects a neighboring solution that has a better quality according to some objective function or evaluation criterion. This process continues until no further improvements can be made, or a stopping criterion is met.

Steps of the Hill Climbing Algorithm:

- Initialization: The algorithm starts with an initial solution, which can be generated randomly or by some heuristic method.
- Evaluation: The current solution is evaluated using an objective function or evaluation criterion, which quantifies the quality of the solution.
- Neighbor Generation: The algorithm generates neighboring solutions by making small modifications to the current solution. These modifications can include changing the values of variables, adding or removing components, or applying local transformations.
- Selection: Among the neighboring solutions, the one that has the highest quality according to the objective function is selected as the next candidate solution.
- Termination: The process continues iteratively until a stopping criterion is met. This criterion can be a maximum number of iterations, reaching a predefined threshold of improvement, or encountering a local optimum.

2.3. SDD-1 Algorithm

The System for Distributed Database (SDD-1) Algorithm is a pioneering approach developed for distributed database systems, aiming to enhance efficiency and performance in accessing and managing data distributed across multiple sites. Developed in the late 1970s, SDD-1 introduced innovative techniques for distributed query processing and optimization, laying the groundwork for subsequent research in this field.

At its core, the SDD-1 Algorithm focuses on minimizing network traffic and communication overhead by executing query operations locally whenever possible, leveraging the data distribution across distributed sites. The algorithm employs a decentralized architecture, where each site autonomously processes local queries and collaborates with other sites as needed to fulfill distributed queries.

Key features of the SDD-1 Algorithm include:

- Query Decomposition: The algorithm decomposes complex queries into subqueries that can be processed independently at distributed sites. This decomposition minimizes the need for data transfer between sites and reduces query response time.
- Local Query Optimization: SDD-1 prioritizes local query execution by optimizing query plans at each distributed site. Local optimization techniques aim to exploit available indices, statistics, and processing capabilities to minimize resource usage and improve query performance.
- Global Query Coordination: While emphasizing local query processing, SDD-1 also incorporates mechanisms for global query coordination and result integration. Distributed sites exchange intermediate results and coordinate query execution strategies to ensure consistency and completeness in query results.
- Dynamic Load Balancing: The algorithm dynamically adjusts query processing strategies based on the current workload and resource availability at distributed sites. Load balancing techniques aim to distribute query processing tasks evenly across sites to avoid bottlenecks and maximize resource utilization.
- Fault Tolerance: SDD-1 incorporates fault tolerance mechanisms to handle failures and ensure system reliability. Redundancy and replication techniques are employed to maintain data availability and consistency in the event of site failures or network partitions.

Algorithm

```
Input: $QG$: query graph with $n$ relations; statistics for each relation
Output: $ES$: execution strategy
begin
    ES $\leftarrow$ local-operations ($QG$) ;
    modify statistics to reflect the effect of local processing ;
    BS $\leftarrow \phi$;                                    {set of beneficial semijoins}
    for each semijoin $SJ$ in $QG$ do
        if $cost(SJ) < benefit(SJ)$ then
            $BS \leftarrow BS \cup SJ$
    while $BS \neq \phi$ do
                                                        {selection of beneficial semijoins}
        $SJ \leftarrow most\_beneficial(BS)$;   {$SJ$: semijoin with $max(benefit - cost)$}
        $BS \leftarrow BS - SJ$;                                  {remove $SJ$ from $BS$}
        $ES \leftarrow ES + SJ$;                       {append $SJ$ to execution strategy}
        modify statistics to reflect the effect of incorporating $SJ$ ;
        $BS \leftarrow BS -$ non-beneficial semijoins ;
        $BS \leftarrow BS \cup$ new beneficial semijoins ;
    {assembly site selection}
    $AS(ES) \leftarrow$ select site $i$ such that $i$ stores the largest amount of data after all
    local operations ;
    $ES \leftarrow ES \cup$ transfers of intermediate relations to $AS(ES)$ ;
    {postoptimization}
    for each relation $R_i$ at $AS(ES)$ do
        for each semijoin $SJ$ of $R_i$ by $R_j$ do
            if $cost(ES) > cost(ES - SJ)$ then
                $ES \leftarrow ES - SJ$
end
```

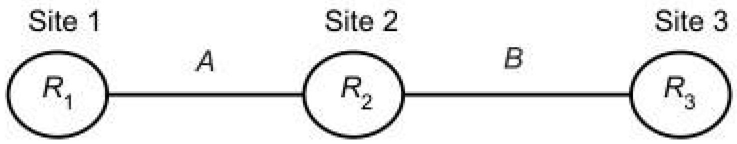
Let us consider the following query:

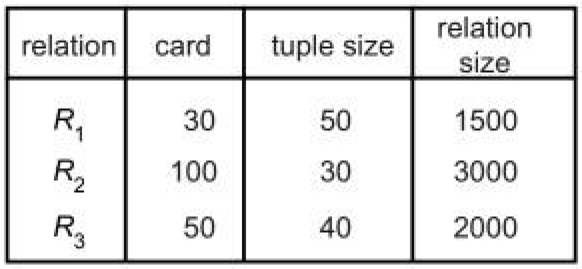
SELECT R3.C

FROM R1,R2,R3

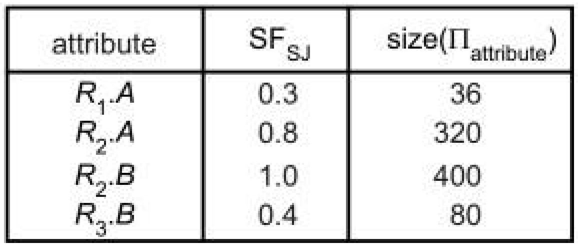
WHERE R1.A = R2.A

AND R2.B = R3.B





relation	card	tuple size	relation size
R_1	30	50	1500
R_2	100	30	3000
R_3	50	40	2000



attribute	SF_{SJ}	$size(\Pi_{attribute})$
$R_1.A$	0.3	36
$R_2.A$	0.8	320
$R_2.B$	1.0	400
$R_3.B$	0.4	80

gives the join graph of the query and of relation statistics. We assume that TMSG = 0 and TT R = 1. The initial set of beneficial semijoins will contain the following two:

SJ_1: $R_2 \ltimes R_1$, whose benefit is $2100 = (1-0.3) * 3000$ and cost is 36
SJ_2: $R_2 \ltimes R_3$, whose benefit is $1800 = (1-0.4) * 3000$ and cost is 80

Furthermore there are two non-beneficial semijoins:

SJ_3: $R_1 \ltimes R_2$, whose benefit is $300 = (1-0.8) * 1500$ and cost is 320
SJ_4: $R_3 \ltimes R_2$, whose benefit is 0 and cost is 400.

At the first iteration of the selection of beneficial semijoins, SJ1 is appended to the execution strategy ES. One effect on the statistics is to change the size of R2 to 900 = 3000 ∗ 0.3. Furthermore, the semijoin selectivity factor of attribute R2.A is reduced because card(ΠA(R2)) is reduced. We approximate SFSJ (R2.A) by

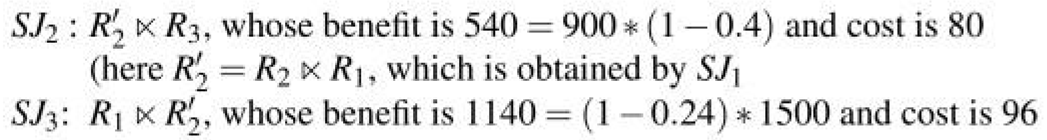
0.8 ∗ 0.3 = 0.24. Finally, size of ΠR2

.A is also reduced to 96 = 320 ∗ 0.3. Similarly,

the semijoin selectivity factor of attribute R2.B and ΠR2.B should also be reduced

(but they not needed in the rest of the example).

At the second iteration, there are two beneficial semijoins:



SJ_2 : $R'_2 \ltimes R_3$, whose benefit is $540 = 900 * (1 - 0.4)$ and cost is 80
(here $R'_2 = R_2 \ltimes R_1$, which is obtained by SJ_1
SJ_3: $R_1 \ltimes R'_2$, whose benefit is $1140 = (1 - 0.24) * 1500$ and cost is 96

, whose benefit is 1140 = (1−0.24) ∗ 1500 and cost is 96.

The most beneficial semijoin is SJ3 and is appended to ES. One effect on the statistics of relation R1 is to change the size of R1 to 360(= 1500 ∗ 0.24). Another effect is to change the selectivity of R1 and size of ΠR1At the third iteration, the only remaining beneficial semijoin, SJ2, is appended to ES. Its effect is to reduce the size of relation R2 to 360(= 900 ∗ 0.4). Again, the

statistics of relation R2 may also change. After reduction, the amount of data stored is 360 at site 1, 360 at site 2, and 2000 at site 3. Site 3 is therefore chosen as the assembly site. The post optimization does not remove any semijoin since they all remain beneficial. The strategy selected is to send (R2 nR1)nR3 and R1 nR2 to site 3, where the final result is computed. Like its predecessor hill-climbing algorithm, the semijoin-based algorithm selects locally optimal strategies. Therefore, it ignores the higher-cost semijoins which would result in increasing the benefits and decreasing the costs of other semijoins. Thus this algorithm may not be able to select the global minimum cost solution.

3. Conclusion

In conclusion, the System for Distributed Database (SDD-1) Algorithm stands as a significant milestone in the evolution of distributed database systems, pioneering innovative approaches to query processing, optimization, and coordination in distributed environments. Developed in the late 1970s, SDD-1 introduced decentralized architectures and dynamic query optimization techniques that laid the groundwork for subsequent research in this field.

SDD-1 emphasized the importance of minimizing network traffic and communication overhead by prioritizing local query execution and leveraging data distribution across distributed sites. Through query decomposition, local query optimization, and global query coordination mechanisms, SDD-1 aimed to improve query performance, minimize response time, and enhance system scalability.

Furthermore, SDD-1 incorporated fault tolerance mechanisms to ensure system reliability and resilience in the face of failures or network partitions. By introducing redundancy, replication, and dynamic load balancing techniques, SDD-1 sought to maintain data availability and consistency, even under adverse conditions.

While subsequent research has introduced more advanced algorithms and techniques for distributed query processing, SDD-1 remains a seminal contribution in the field of distributed databases, shaping the direction of research and inspiring further innovation. Its principles and methodologies continue to influence the design and implementation of distributed database systems, providing valuable insights into the challenges and opportunities of managing data in distributed environments.

4. References

Principles of Distributed Database Systems

Book by M. Tamer Özsu