2002

# Dynamic techniques in distributed query optimization.

Lubna. Sachwani
*University of Windsor*

# Dynamic Techniques in Distributed Query Optimization

by

**Lubna Sachwani**

A Thesis
Submitted to the Faculty of Graduate Studies and Research
Through the School of Computer Science in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the University of Windsor

Windsor, Ontario, Canada
2002

**Lubna Sachwani**

## ABSTRACT

A need for effective systems and tools for information retrieval and analysis is becoming more apparent with the continuous and rapid growth of information sources and information users. Query Optimization, thus, is a key research area in Database Management Systems. A new JAL dynamic algorithm is presented. This is an efficient method of creating the query plans dynamically without having to provide extra costly information to help decision-making. With no monitoring required, the query plans are generated on the fly and the query response time is minimized. This heuristic is not only dynamic but it also ensures that the query response time is either less than or equal to the existing static query plans without increasing the complexity of the query processing.

## DEDICATIONS

I would like to dedicate my thesis to one great person, who has been an immense support to not only me but to my family as well. He and his family will be constantly in my (and my family's) prayers and love forever. He has been more than a father to me and I pray to Almighty Lord to fill his life and the lives of his family members with His infinite blessings and joy - Ameen.

**ACKNOWLEDGEMENTS**

I want to express my immense gratitude to my dearest Supervisor, Dr J. M. Morrissey, who has always been there for me. Ever since I started my Master's program, she has been extremely kind and has supported me always. Her guidance has helped me accomplish my goals; her smiling face has always welcomed me and her warmth has always made me at home. She has not only been my Supervisor, but also as an elder has helped me see things from different perspective – and this helped me grow and learn. She is one of those people, who I admire the most.

At this point of time, I want to recognize all my professors and teachers especially Dr Kent for his continuous support. I am not mentioning any other names because I do not want to miss out anyone as from each one of them I have always learnt something. Aristotle remarked:

"My parents brought me from the sky to the earth, but My Teacher will take me from the earth to the sky."

I will always be indebted to all my teachers and my thesis committee members (in alphabetical order): Dr Dickinson, Dr Morrissey, Dr Ngom and Dr Tsin for their guidance and a lot of support. Their invaluable advices gave my thesis its professional look. Not to forget Dr Dickinson, who was my very first professor at the University of Windsor in the School of Business, and who also welcomed me on my very first day of school in Windsor.

My extreme gratitude goes once again to that great admirable person; who is very dear to me; who helped me achieve my goals; whose unending love and support will always remain with me forever; who I always pray for and who I will never ever forget. And I have also dedicated my thesis to him. I just have to say: " 'Allah creates whatever He wants whenever He wants.' Only Allah makes the strongest relationships using the most

fragile threads and only Allah fills the heart with love. You are also my most beloved father."

I can never forget the love, the support, the strength, the guidance, the help, the right upbringing, the right education and moral values given to me by my beloved parents, brother and aunty. I will never be able to make it to them for what they have done for me. I just want to say, "Without you all, I do not exist."

"It is choice, not chance, that determines our destiny." - Jean Nidetch
He who is within me and with Him I exist: Almighty Allah. I want to give my extreme humble "Shukrana" to God. He helped me make the choices and His love nurtured me. He is the most beneficent and the most merciful.

I want to pray to God: "O Lord, give all my benefactors and myself that insightful vision that we may love you and obey your commands and fill our lives – physical and spiritual – with your infinite blessings and light - Ameen."

"Knock, and He'll open the door. Vanish, and He'll make you shine like the sun. Fall, and He'll raise you to the heavens. Become nothing, and He'll turn you into everything." - Rumi.

## TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

# 1. INTRODUCTION: DISTRIBUTED QUERY OPTIMIZATION

## 1.1 INTRODUCTION

A need for effective systems and tools for information retrieval and analysis is becoming more apparent with the continuous and rapid growth of information sources and information users.

With the technology growing at an exponential rate, "clicks" becoming more powerful, the world becoming a "global village," and we living in the "Information age," one is motivated into finding out ways to achieve the faster query results with minimal cost and without overloading the networks and excessive intersite data transfer.

Undoubtedly, distributed database management systems have some unique advantages over the traditional centralized database management systems. These advantages include: better performance, reliability, availability and better modularity, just to name some.

The distributed database management system (DDBMS) is a collection of independent computers connected via point-to-point communication lines [KYY+83]. The computers, also known as nodes, are autonomous in the sense that each one has its own storage, I/O and processing capabilities. The data in the distributed environment is distributed across the network. The DDBMS provides a unified interface to the users, so that they can access the database without knowing the internal processing.

To the users the data querying would appear as if they were querying a local database. Therefore, the foremost objective in DDBMS is to establish methods, rather query optimization methods, to achieve the task with minimal time. What is of importance to businesses, and investors in particular, is to have fast and reliable access to the information, which by the use of reliable query processing techniques can be accomplished.

Query optimization has always been one of the most crucial components of database technology. Its goal is to achieve efficient processing of users' queries that retrieve data

from an often very large database. There has been extensive work in query optimization since the early 1970s [Won77]. However, today, centralized/distributed relational database systems can no longer meet the ever-growing demand of database support from diversified applications. In response to the demand, researchers and practitioners in the database community have continuously put forward new ideas for developing new systems and enhancing existing ones. As a result, many advanced database systems have been emerging. These new types of database systems have raised many new challenges for query optimization. However, this thesis mainly focuses on the relational database systems. The main concern is to retrieve the data that is spread across different sites in the least possible time with the minimum cost. The retrieval of data from different sites is called distributed query processing [ES80].

One essential point to consider is that there exists Network Transparency. This means that the users are shielded from the low level details such as how replicas or the fragments are distributed. The user should be able to carry out the queries in the same way as if he or she is querying a local database.

A lot of work has been done to optimize the queries. Many algorithms have emerged, thus giving the DDBMS the option to chose to execute the query. Eugene Wong of the University of California at Berkeley did the initial work on query optimization [Won77]. He proposed an optimization method based on a greedy heuristic that produces efficient but not necessarily optimal query processing strategies. An enhanced version of this method is implemented in the SSD-1 system. When we consider DDB, we have a number of issues to take into consideration.

One of the major concerns is the increased complexity of the distributed systems. This implies greater development costs and thereby, greater potential for coding errors. Moreover, we also have to take into account the increased overheads due to message passing and data transfers. Another crucial issue is the security. However, undoubtedly the advantages of having a DDBMS outweigh its disadvantages.

We also have to take into account the design issue. In the general setting, it includes making decisions about the placement of data and programs across the sites of a computer network as well as possibly designing the network itself. In DDBMS, the placement of applications entails:

1. The placement of the distributed DBMS software and,
2. The placement of the applications that run on the database.

The question pertaining to the design issues now is how to distribute the relations. Relations can be fragmented. A fragment of a relation is a sub-relation. A relation is fragmented in such a way that the original relation can be reconstructed without the loss of any information. Two common ways of fragmenting are horizontal fragmentation and vertical Fragmentation. We also have to take into account the replication of these fragments. A distributed system can be non-replicated implying each fragment resides at only one site. It could also be partially or fully replicated. The advantages of replication are obvious: It allows parallel processing and minimizes data transfer costs. However, there are huge overheads in terms of update. This is beyond the scope of this thesis work.

In 1983, Aper, Henver and Yao (AHY83) proposed an algorithm called Algorithm General, which mainly focused on:

1. Minimize response time.
2. Minimize the total time (i.e., minimize total data transfer).

Algorithm General originated from two algorithms: Parallel and Serial. Algorithm parallel was originated to reduce the response time and algorithm serial to reduce total time.

This is regarded as the best heuristic to date. Other techniques to reduce query response time have also been proposed such as two-way and n-way semijoins by Kang and Roussopoulos [KR87] [RK91]. Perrizo et al worked on composite semijoins [PC90].

Another kind of semijoin operator used to reduce the cost and thus optimize queries is called the Hash semijoin [TC92].

Algorithms for processing distributed queries require a priori estimates of the size of intermediate relations [BRP92]. In 1992, P. Bodorik, J. Riordan, and J. Pyra [BRP92] coined the idea for making query optimization dynamic. Most such algorithms take a "static" approach in which the algorithm is completely determined before processing begins. If size estimates are found to be inaccurate at some intermediate stage, there is no opportunity to reschedule, and the result may be far from optimal. Adaptive query execution may be used to alleviate the problem. Care is necessary, though, to ensure that the delay associated with rescheduling does not exceed the time saved through the use of a more efficient strategy. Their paper presents a low overhead delay method to decide when to correct a strategy. Sampling is used to estimate the size of relations and alternative heuristic strategies prepared in a background mode are used to decide when to correct. Correction is made only if lower overall delay is achieved, including correction time. Evaluation using a model of a distributed database indicates that the heuristic strategies are near optimal. Moreover, it also suggests that it is usually correct to abort creation of an intermediate relation, which is much larger than predicted.

Taking into account the idea of making query optimization dynamic, this thesis proposes a new algorithm, the JAL dynamic algorithm, based on the heuristic of AHY [AHY83]. This thesis examines the following questions and proposes the solutions to them:

1. Can the dynamic approach be used to improvise the best of the heuristics existing today?

2. What can be done to ensure that the proposed algorithm achieves the same results with either less than or equal to the response time achieved by the heuristic?

3. What would be the impact on the complexity of the algorithm by using this approach?

In order to answer these vital questions, this thesis compares the AHY algorithm to the proposed JAL dynamic algorithm and tests the results to ensure that the objectives are achieved.

## 1.2 ORGANIZATION OF THE THESIS

This thesis consists of six chapters. Chapter 1 gives a brief introduction to distributed query optimization. Chapter 2 gives the relevant background information. This includes various heuristics algorithms, join techniques, and static and dynamic distributed query optimization. Chapter 3 discusses the key concepts: stages in distributed query optimization, assumptions, definitions and notations and the query graphs. In chapter 4 a detailed description of the AHY and JAL-dynamic heuristics is given.

Chapter 5 discusses the evaluation methodology. The benchmark database and the relevant queries are discussed. It also includes the testing and experimental results. Chapter 6 states the conclusion after evaluating the results. Future work is also discussed in this chapter.

# 2. BACKGROUND

The previous chapters focused on the need for the optimization for the queries and key concepts along with some important definitions related to distributed query optimization respectively. This chapter provides background information on the problem dealt within the thesis. The specific query optimization objectives are discussed followed by some heuristics adopted to overcome the problems and make querying both more efficient and effective. An in depth look is taken at some strategies.

## 2.1 OBJECTIVES OF THE DISTRIBUTED QUERY OPTIMIZATION

Some specific distributed query optimization objectives include reduction of [OV91] [Mor01]:

1. The volume of the data transferred over the network.
2. The size of the partial results. A partial result refers to the size of the relation after the application of a relational operator.
3. The dollar cost of both the local CPU processing and network processing. Either in terms of monitoring or the provision of extra data to foster decision-making.
4. Delay due to both local CPU and network data transfers.
5. Volume of data processed by all information processors.

## 2.2 OPTIMIZATION TECHNIQUES

When a program written in a procedural language, for example Pascal or C, is to be executed, a compiler first translates the program into machine language usually. The translation involves taking the source code and generating equivalent machine code that preserves the sequence of operations that the source code specifies. As part of the translation process, the compiler carries out code optimization to generate as efficient a code as possible. This might include elimination of unnecessary writing of values to the memory and reading them to the registers, some clever unrolling of the loops, etc.

In a non-procedural or a declarative language like SQL, no sequence of operations is explicitly given and therefore a query may be processed in a number of different ways (often called **plans**) where each plan might have a different sequence of operations.

Optimization in non-procedural languages therefore is a much more complex task since it not only involves code optimization (how to carry out the operations that need to be carried out) but also selecting the best plan as well as selecting the best access paths. In many situations, especially if the database is large and the query is complex, a very large number of plans are normally possible and it is then not practical to enumerate them all and select the best. Often, then, it is necessary to consider a small number of possible plans and select the best option from those. Since the savings from query optimization can be substantial, it is acceptable that a database spends time in finding the best strategy to process the query. Finding the best execution algorithm or the query plan is NP-hard [CWY88]. Again, of course, we assume that we are dealing with queries that are expected to consume a significant amount of computing resources; there is no point in spending time optimizing queries that are so simple that any approach can be used to execute them in a small amount of time.

Chen and Yu in [CY90a] [CY90b] proposed a theorem to estimate the expected number of tuples in a relation:

**Theorem:** Let G= (V, E) be a join query graph and $G_R = (V_R, E_R)$ is a connected graph of G. Let $R_1, R_2, \ldots, R_p$ be the relations corresponding to the nodes $V_R$, and $A_1, A_2, \ldots, A_q$ be the distinct attributes associated with edges ER. Let $n_i$ be the number of different nodes (relations) that edges with attribute Ai are incident to. Suppose $R^*$ is the relation resulting from the join operations between $R_1, R_2, \ldots, R_p$ and m is the expected number of tuples in $R^*$. Then,

$$m = \frac{\prod_{i=1}^{p} |R_i|}{\prod_{j=1}^{q} |A_j|^{nj-1}}$$

Suppose we have the following data of the attributes as shown below:

| RELATION | JOINING ATTRIBUTE |
|---|---|
| R1 | A, B, C |
| R2 | A, B, D |
| R3 | A, C |
| R4 | A, D |

**Table 2.1: Relations and their joining attributes**

The expected number of tuples using the above theorem will be:

$$m = \frac{\prod_{i=1}^{4} |R_i|}{|A|^3 |B||C||D|}$$

## 2.3 METHODS OF QUERY OPTIMIZATION

### 2.3.1 INITIAL FEASIBLE SOLUTION (IFS)

As the name suggests IFS is a very naïve way of retrieving the results. Basically in IFS we ship all the relations directly to the query site and perform joins there. It is very simple but rarely efficient [AHY83].

### 2.3.2 USING JOINS AS REDUCERS

Suppose we have two relations R1 and R2 such that R1 is larger in size then R2 and both have a common joining attribute A [ME92]. Simply what we do is to ship R2 to the site of R1 and perform a join there. Ship the result of the query after the join to the query site.

An example is given below:

**R1**

| A | B |
|---|---|
| 1 | 5 |
| 2 | 6 |
| 3 | 7 |
| 4 | 8 |

**R2**

| A | D |
|---|---|
| 1 | 8 |
| 2 | 69 |

**Table 2.2: Relations R1 and R2**

In the IFS we ship both R1 and R2 to the query site. In the Join strategy, we ship R2 to R1, perform join at R1 and ship the final result to the query site thus reducing the extra data transfer of R1.

However, sometimes joins can be very expensive depending on the kind of relation as illustrated below:

**R1**

| A | B |
|---|---|
| 1 | 5 |
| 1 | 7 |

**R2**

| A | D |
|---|---|
| 1 | 8 |
| 1 | 4 |

**Table 2.3: Expensive Join Relations R1 and R2**

In this case the IFS is less costly than the Join strategy.

### 2.3.3 SEMIJOINS

As described earlier, the objective of query optimization is to minimize the cost of intersite data transfer. Also, joining a whole relation to another costs more than if we just perform a semijoin. The semijoin notation is as follows:

Ri $\rtimes$ Rj and some authors have also used Ri – k → Rj to express the same.

The following are the steps involved in semijoin [ME92]:

1. Project relation Ri over attribute k to get the projection dik.
2. Ship the projection dik to the site of relation Rj, which is larger in size compared to Ri.
3. Execute dik $\bowtie$ Rj.

Therefore, we can say that a semijoin is equivalent to the composition of join and projection. It is not commutative unlike other join operations [MB96]. So

$(Ri \rtimes Rj) \neq (Rj \ltimes Ri)$

The cost of a semijoin is a linear function of the volume of the joining column to be transferred as given below:

$C(Ri \ltimes Rj) = C1\ |dja| + C2\ |dja \bowtie Ri| + C0$

Where, C0 is the start-up cost of initiating transmission and C1 and C2 are the network-dependent constants. The following is an example of a simple semijoin algorithm: Suppose we have relations R1 and R2 with the joining attribute as A.

**R1**

| A | B |
|---|---|
| 1 | 5 |
| 2 | 6 |
| 3 | 7 |
| 4 | 8 |

**R2**

| A | D |
|---|---|
| 1 | 8 |
| 2 | 69 |

**Table 2.4: Relations R1 and R2**

In the case of the semijoin, we will project R2 over A as we can see that the size of R2 is lesser than the size of R1. After projection we will send A costing us 2 to R1, where the join will be performed. This in fact, is the semijoin and the total cost is 4, cost of the attribute B. And if need be, the attribute D can also be shipped to the query retrieval site. Therefore, our cost is reduced significantly.

### 2.3.4 USING BLOOM FILTER

A Bloom filter is a simple space-efficient randomized data structure for representing a set that supports approximate membership queries.

The Bloom Filter algorithm was originally documented in the Communications of the ACM in 1970 (Vol. 13(7)) by Burton H. Bloom. Bloom filters are increasingly being used in distributed systems; for example, in a distributed Web cache, proxies do not need to share the exact URL lists of their caches, but instead periodically broadcast Bloom filters representing their cache.

Suppose we have Ri and Rj and we intend to perform a semijoin on this. Then we will send the projection of Ri on the joining attribute A to Rj to perform the join. In the case of the hashing, we use the search filter, which represents the semijoin projection with a small bit array. This filter is called Bloom Filter as it was introduced by Bloom. This technique provides an alternative way to reduce the local processing load of regular relational operations and thus avoid the creation of large intermediate relation results for transmission.

We have two competitive approaches to estimate the size of the relational approach [Mul93]. They can by classified according to the way in which each individual bit is addressed. Therefore, there are two variations on bit vectors [Mul93]. In the value vector, it is directly addressed by the key attribute values from a participating relation or some compact representation of the key attribute values. However, one has to take into consideration the fact that if the selected tuples are a large number because of the

selection predicate, then it is not cost effective. This method is more effective for relations that have not been already reduced.

The other is the hashed based filter and this is very widely used. This method works well even when the original relations have been reduced. One advantage of using Bloom filters is that one scan of the relation is enough to build the filter and then only a thousand of bit storage space is required. Hashing the attribute values to a smaller range of addresses reduces the size of the Bloom Filter. Just like hashing, the hash function is applied to a key or an attribute in the case of the relation, to produce the address in the data structure. The Bloom filter is nothing but a bit array. It is a compact representation of join attribute values.

Suppose we have a relation A given below:

**A**

| Supplier Number (Primary Key) | Supplier Name |
|---|---|
| 2 | Adams |
| 3 | Smith |
| 5 | Kelly |
| | |
| | |

**Table 2.5: Relation A**

The filter for this can be constructed as follows [Mor01][TC92]:

1. Initialize to zero
2. For each value, hash to produce address
3. Set address to 1

Therefore, the filter would be as follows:

| |
|---|
| 0 |
| 1 |
| 1 |
| 0 |
| 1 |
| 0 |

| 0 |
|---|
| 0 |
| 0 |
| 0 |
| 0 |

**Table 2.6: Filter for Relation A**

Now to join relation A to Relation A1, on a joining attribute, e.g., Supplier Number, the steps are as follows:

1. Apply the hash function to Supplier Number in A1
2. For each address produced in 1 test for the presence of a 1 bit in h(b)
3. If the address is same as the relation A then that tuple is kept
4. The further processing is done on 3 and the rest are discarded.

This is shown as follows for the Relation A1 [Mor01].

| Supplier Number | Part Number | Price |
|---|---|---|
| 1 | 20 | 100 |
| 1 | 22 | 109 |
| 1 | 23 | 128 |
| **2** | **20** | **101** |
| **2** | **24** | **123** |
| **3** | **25** | **140** |
| **3** | **26** | **170** |
| **3** | **27** | **158** |
| 4 | 20 | 150 |
| **5** | **21** | **145** |
| **5** | **25** | **134** |
| 6 | 28 | 128 |

**Table 2.7: Relation A1 (note the tuples in the bold are kept)**

The approach given in [TC92] to use hash semijoin assumes that we have an execution graph of the joining attribute. This implies that it can only work with tree queries, which are a small fraction of the queries. However, once we have a query graph then it is a very

cost effective way of performing joins. The authors state that work on devising the algorithm for general queries or general execution graphs is still under investigation.

One thing to note with hash-semijoin is the problem of collision. However, collision here implies that one or two key attributes are mapped to the same address. This will give us some extra tuples. It guarantees that no required tuples will be lost. Also, we can apply multiple filters concurrently. Using filters also implies that there is no "delete a member" operation available on the filter. If a join key is removed from a file in an update, then one must either live with the inaccuracy of the filter or completely rebuild the filter [Mul93].

## 2.4 ALGORITHMS USING SEMIJOINS

Many algorithms have been developed such as composite semijoin, two-way semijoin, n-way semijoin, composite semijoin and domain specific semijoin to name a few. Some are explained briefly below:

### 2.4.1 SDD-1 ALGORITHM

SDD-1 was a distributed database system developed by the Computer Corporation of America. It permits a relational database to be distributed among the sites of a computer network. Users interact by submitting queries in a high-level procedural language called Datalanguage. The paper gives the techniques to optimize queries in SDD-1. The process of optimization starts with translating each datalanguage query into relational calculus. This is also called "Envelope."

The objective of the algorithm described in this paper is to minimize the quantity of intersite data transfer. Therefore, it assumes that the network bandwidth is the bottleneck and all the resources are then free [BWG+81].

The algorithm has the following three steps:

1. Form the envelopes
2. Evaluate the envelopes and assemble distributed data at assembly site

3. Local processing of the datalanguage query.

Initially using select, project operations reduce the size of a relation. Thus, we are reducing the size of the data transfer. Then, we compute join operation. The algorithm uses semijoin. The paper gives the reasons for using semijoin:

1. Semijoins monotonically reduce the size of the database compared to join operation where the cost can increase the size of the database in the worst case.
2. Semijoins can be computed with least intersite data transfer.

The algorithm OPT is reiterated as follows [BWG+81]:

1. Initialize the program, P, with all the local operations permitted by the envelope E. E is the relation calculus, which reduces the database to its subset.
2. Check if the local joins in E are profitable. If yes, append to P.
3. While no profitable semijoins exist Do step 2.
4. Select an assembly site, Sa, to be the site with maximum data size.
5. Append to P commands to move data from all other sites to Sa.

OPT is an example of a greedy algorithm. The paper recognizes the fact that this algorithm always seeks to maximize the immediate gains. It does not look ahead nor does it back up. As a consequence, the reducers generated by OPT are suboptimal. It therefore also is considered as a hill-climbing approach.

However, there is also a lot of communication cost involved in transmitting the information such as what joins should be performed and what is the assembly site.

### 2.4.2 AHY ALGORITHM

Essentially we are concerned with the efficiency of processing strategies for queries in a distributed database for system performance. A new algorithm called Algorithm General achieves this objective [AHY83] by:

1. Minimizing response time.
2. Minimizing the total time (i.e., minimize total data transfer).

Algorithm General originated from two algorithms, Parallel and Serial. Algorithm parallel was originated to reduce the response time and algorithm serial to reduce total time. The following is a brief outline of both the algorithms:

"Algorithm PARALLEL

1. Order relations $R_i$ such that $S_1 <= S_2 <= \ldots <= S_m$ (Si: Size of $R_i$)
2. Consider each relation $R_i$ in ascending order of size
3. For each relation $R_j$ (j<i), construct a schedule to $R_i$ that consists of the parallel transmission of the relation $R_i$ and all schedules of relations $R_k$ ($k < j$). Select the schedule with minimum response time.

"Algorithm SERIAL

1. Order relations $R_i$ such that $S_1 <= S_2 <= \ldots <= S_m$ (Si: Size of $R_i$)
2. If no relations are at the result node, then select strategy: $R_1$ - >$R_2$ ->...->$R_n$-> result node.

Or else if $R_r$ is a relation at the result node, then there are two strategies:

$R_1 \rightarrow R_2 \rightarrow R_r \rightarrow \ldots \rightarrow R_n \rightarrow R_r$ or

$R_1 \rightarrow R_2 \rightarrow \ldots \rightarrow R_{r-1} \rightarrow R_{r+1} \rightarrow \ldots \rightarrow R_n \rightarrow R_r$.

Select the one with minimum total time." pp. 60-61 [AHY83].

Algorithm Parallel and Serial were extended to Algorithm G to achieve the objectives. However, it had some problems. Therefore, Hevner and Yao [AHY83] recognized these problems and developed improved Algorithm GENERAL.

There are three versions of Algorithm GENERAL.

For Response Time: **Procedure RESPONSE**

For Total Time: **Procedure TOTAL** and **Procedure COLLECTIVE**

A fundamental assumption for this algorithm is that it is applied to only simple queries. The data transmissions used for reducing a relation and the transmission of the reduced relation to the query computer form a **schedule** for this relation. The response time of a schedule is the time between the start of the first transmission and the time at which the relation (attribute) arrives at the required computer.

The minimum **response time of a relation** (attribute) is the minimum response time among all possible schedules for this relation (attribute). The **total time of a schedule** is the sum of the costs of all transmission required in the schedule.

Suppose a query requires us to join three relations $R_1$, $R_2$ and $R_3$ such that we need to project the values of attributes 1 and 2. First the projection of attribute 1 from $R_2$ is sent to $R_3$, where the semijoin is performed. The reduced relation at $R_3$ is sent to $R_1$ where the final join is performed. Concurrently the projection of attribute 2 from $R_2$ is also sent to $R_1$ where the semijoin is performed once again for attribute 2. The final result is sent to the query site or the result node. This is shown in the figure below:

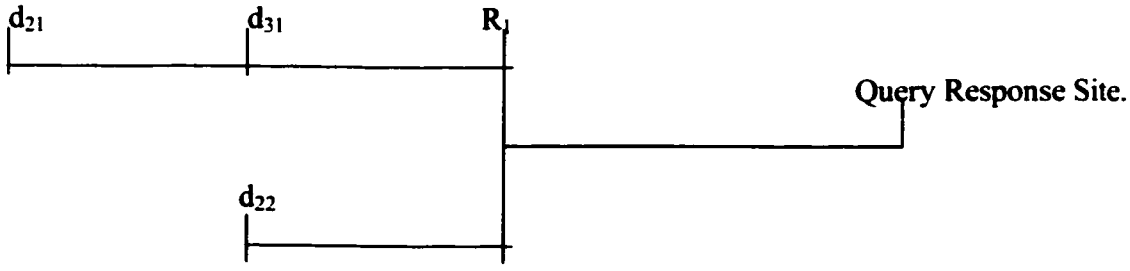



**Figure 2.1: pp. 58 [AHY83]**

"Algorithm General

1.Do all initial local processing

2.Generate candidate relation schedules

a) To minimize response time, use **Algorithm PARALLEL** to each relation (simple query)

b) To minimize total time, use **Algorithm SERIAL** to each simple query.

3. Integrate the candidate schedules

- For each relation $R_i$, the candidate schedules are integrated to form a processing schedule for $R_i$ . The integration is done by **Procedure RESPONSE** for response time minimization and by **Procedure TOTAL** or **Procedure COLLECTIVE** for total time minimization.

4. Remove schedule redundancies." pp. 60-61 [AHY83].

"Procedure Response

1. Candidate schedule ordering: the schedules of attributes that can be applied to relation $R_1$ are ordered on their arrival time in the $R_1$ node.

2. Schedule Integration: For each candidate schedule $CSCHL_k$ in ascending order, construct an integrated schedule for R that consists of the parallel transmission of $CSCHL_1$
and $CSCHL_k$ with $k < l$. Select the integrated schedule with minimum response time.

"Procedure Total

1. Adding candidate schedules. For each relation Ri and each candidate schedule CSCHL, do the following:
   If this schedule contains a transmission of a joining attribute of Ri, say dij, then add another candidate schedule that is the same as CSCHL except that the transmission of dij is deleted.
2. Select the best candidate schedule. For each relation Ri and for each joining attribute dij (j=1, 2, ..., m) select the candidate schedule which minimizes the total time for transmitting Ri if only the joining attributes are considered which can be joined with dij.

3. Candidate schedule ordering. For each relation Ri, order the candidate schedules BESTij on joining attributes dij (j=1, 2, ..., m) so that Arrival Time is the least of the BESTij.
4. Schedule integration for each BESTij in ascending order of j, construct an integrated schedule to Ri that consists of the parallel transmission of candidate schedule BESTij and all schedules BESTik where k<j. Select the integrated schedule that results in the minimum total time value.

"Procedure Collective:
This procedure is not an optimal solution as it considers the transfer of redundant data in the relations. It is as follows:

1. Select candidate schedule
2. Build processing strategy
3. Test variations of strategy." pp. 60-61 [AHY83].

### 2.4.3 STRATEGY V

An algorithm called Strategy V [VYV+84] is an improvement to the algorithms Method-D [CY80] and Algorithm General [AHY83]. The AHY algorithm does not take into consideration the cost of local processing. However, this algorithm takes into account the local processing costs. Therefore, the cost for R1 $\bowtie$ R2 is defined as:

$$C2(R1, R2) = V0 + V1 * \frac{|R1| + |R2| + |R0|}{\text{Block}} + V2 * \frac{|R1| + |R2|}{\text{Byte}}$$

Where,
Block = transfer rate (block/sec)
Byte = transmission rate (average bytes/sec)
V0 = Setup cost
V1 = Weight associated with block access
V2 = Weight associated with CPU usage

Therefore,

$C(R0) = C2(R1, R2) + C0 + C2 * R0$

The above cost includes both the communication and the local processing costs.

Query optimization steps for the Strategy V are:

1. Relations are ordered and labeled and their initial schedules are derived. Two possibilities taken into account are: The direct transmission of a relation and semijoining a relation with another relation residing at the same site prior to the transmission.

2. Order and label the sites based on their initial schedules. This may have effect on reducing the number of sites involved.

3. Attempts are made to improve the initial schedules by generating candidate schedules. Improved schedules are created considering various alternative schedules involving relations and candidate schedules of relations having only small numeric labels.

### 2.4.4 ONE-SHOT SEMIJOIN EXECUTION

Execution of semijoins in most strategies is sequential. This implies that the reduction affect of one semijoin can be propagated to another and so on [WLC91]. Therefore, the cost of the semijoin execution can be lowered if Ri $\rightarrow$ Rj is performed followed by Rj $\rightarrow$Rk. However, there is a drawback to this strategy.

One essential problem, which arises by executing semijoins sequentially, is the loss of parallelism. Other disadvantages include: processing overheads, loss of global semijoin optimization and inaccurate semijoin reduction estimation.

Wang, Li and Chen [WLC91] addressed this problem by introducing a new method called "one-shot semijoin." All applicable semijoins to the relations are executed at the

same time, implying that the semijoin processing at all sites can be done simultaneously [WLC91]. Therefore, we will have to scan each relation only once to process all the applicable semijoins and there will be no propagation of cost or benefit.

### 2.4.5 COMBINATORIAL OPTIMIZATION

Combinatorial optimization methods were first used in centralized databases systems. They are methods to find a "close to optimal" schedule in solving large distributed query optimization problems. They can be used for optimizing the semijoin schedule that fully reduces all relations of a tree query [SG88]. They are based upon the generation of random points in the domain of the objective function. Techniques to solve these problems include: random search, single start, multi start, simulated annealing and combination of random search and local simulated annealing. These techniques reduce search space significantly step by step taking into account the validity and cost requirements. Finally we can find the optimal semijoin schedule that fully reduces all relations of the tree query. Groselj and Malluhi [BM95] illustrated each query as a combinatorial object, which can be efficiently generated using a stochastic algorithm. It can be defined as follows:

$$f^* = \min_{x \in B} f(x)$$

Where,

B is a discrete finite set, that is, the domain of a combinatorial object x, f:B→R, is the objective function and $f^*$ is the optimal values for f.

### 2.4.6 TWO-WAY SEMIJOINS

The objective of this paper is to minimize data transfer over the network.

Basically the reduction phase in the stages in query optimization has to be optimized. For general queries, finding the optimal solution is NP-hard so the authors in this paper have tried to develop a heuristic which produces an acceptable if not optimal answer.

Most execution phases in query optimization utilize the semijoin operator to reduce the size of the relation. Kang and Roussopoulos [KR87] [RK91] introduced a new relation operator called the two-way semijoin. The two-way semijoin is stated as follows:

$Ri \leftarrow A \rightarrow Rj = \{Ri - A \rightarrow Rj, Rj - A \rightarrow Ri\}$

This means the first Ri is projected onto A and the projection goes to Rj to reduce Rj. From the reduced Rj, which we should call Rj' now, we project A again and send it to Ri to reduce Ri. One thing worth noting is the fact that the Rj, which goes to Ri in the second step, is the reduced Rj and not the original Rj. The authors did not use the symbol effectively.

The following are the steps in two-way semijoin $Ri \leftarrow A \rightarrow Rj$, where $i \neq j$:

1. Project Ri[A] and send it to Rj.
2. Reduce Rj by eliminating the tuples whose attributes A are not matching any of the Ri[A]. This is the forward reduction phase. During this forward reduction phase, Ri[A] is partitioned into Ri[A]m and Ri[A]nm. The Ri[A]m is the set of values in Ri[A] which match Rj and the other is the set of nonmatching values.
3. Send Ri[A]m or Ri[A]nm from site i to j, which ever is smaller.
4. Reduce Ri keeping matching tuples if Ri[A]m was sent or keep the nonmatching tuples if Ri[A]nm was sent.

An example to illustrate the two-way semijoin execution:
Suppose we have relations Ri and Rj with the joining attribute A. The relations are as follows:

**Ri**

| A | C |
|---|---|
| 1 | 9 |
| 2 | 8 |

| | |
|---|---|
| 3 | 7 |
| 4 | 6 |
| 5 | 5 |
| 6 | 4 |
| | |

**Rj**

| A | B |
|---|---|
| 2 | 2 |
| 4 | 4 |
| 5 | 6 |
| 6 | 8 |
| 8 | 10 |
| 10 | 12 |

**Table 2.8: Two-way semijoin Relations Ri and Rj**

The projection of Ri over A is sent to Rj and the following table shows the tuples kept by Rj after the join:

| A |
|---|
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |

**Table 2.9: Ri[A]**

| A | B |
|---|---|
| **2** | **2** |
| **4** | **4** |
| **5** | **6** |
| **6** | **8** |
| 8 | 10 |
| 10 | 12 |

**Table 2.10: Rj (note the bold tuples are kept)**

Here Ri[A]m = (2, 4, 5, 6)

And Ri[A]nm = (1, 3)

In the above case we will send Ri[A]nm to site Ri because the number of the non-matching tuples is smaller. However, if Ri[A]m were fewer we would have sent those to Ri and done the join.

The cost and benefit of two-way semijoin is given as follows:

**Given t: Ri←A→Rj ≡ s: Ri - A→ Rj and $s^r$: Rj - A→ Ri**

Where t = two-way semijoin

And s = semijoin

$s^r$ = semijoin reverse.

**Benefit of t =**

**[S(Ri) – S(Ri')] + [S(Rj) – S(Rj')]**

**Cost of t =**

**S(Ri[A]) + min [S(Ri[A]m), S(Ri[A]nm)]**

The two-way semijoin, t, is profitable when the benefit is higher than the cost.

The paper gives some lemmas and following properties of t:

1. If s reduces Rj to Rj', then t reduces Rj to Rj'.
2. Cost and benefit of reducing Rj by s and t is the same.
3. Ri is always reduced by t in a cost effective way, $s^r$ may not be.

The paper states that if s is cost effective then t is also cost effective and vice versa. However, the above might not be true depending on the selectivity. We might not have a

benefit if the selectivity of Rj is 1, implying no reduction, or Ri has a single attribute. The following example illustrates in detail:

| Relations | Size | dil | Selectivity |
|---|---|---|---|
| R1 | 200 | 100 | 0.5 |
| R2 | 2000 | 1000 | 1 |

**Table 2.11: Relation Data**

Therefore, a two-way semijoin may not be beneficial as shown below:

| Two-way Semijoins |
|---|
| Ri - A→ Rj = 100<br>Rj - A→ Ri = 1000 |
| Ri - A→ Rj = 2000 – (0.5 * 2000) = 1000<br>Rj - A→ Ri = 200 – (1 * 200) = 0 |
| Profit of t = - 100. |

**Table 2.12: Cost and benefit of two-way semijoin**

The paper also discusses the propagation effect of the two-way semijoin and semijoins. It is important to note that the two-way semijoin strategy can be extended to n-way semijoins.

### 2.4.7 COMPOSITE SEMIJOIN

Semijoin specific techniques are classified into four categories: single attribute-semijoining, relation semijoining, composite semijoining and composite relation semijoining. Perrizo et al. worked on composite semijoins [PC90].

Simply stated, the composite semijoin method is applicable whenever there are multiple joining attributes in two relations. A composite attribute semijoin replaces the individual single attribute semijoins. In the combination of composite semijoining and relation semijoining, the entire relation is sent whenever a transmission is made and whenever there are multiple joining attributes. This technique does minimize the response time for the queries involving multiple joining attributes.

The paper has defined the composite semijoin as follows:

"It is a semijoin in which the projection and transmission involve multiple columns." p. 50 [PC90].

A query can require multiple joining between two relations. In that case instead of joining as multiple single columns, we can do semijoin as one composite and this can reduce the response time. The paper states that the new strategy is at least as efficient as the multiple column semijoin strategy. However, if in some situation we find that it is increasing the query response time then we can always use the older strategies. There is no increase in the overhead. The counting process would take a little longer since the size counts for the composite joining attributes would have to be made as well as the size counts for the single joining attributes. However, this is not done at query time.

### 2.4.8 DOMAIN SPECIFIC SEMIJOIN

Chen and Li [CL90] introduced a new operation called domain specific semijoin. Usually when a semijoin is performed it is in the form of relation-to-relation or a relation-to-fragment. Domain specific semijoins exploit the semantic information associated with the joining fragmented relations to reduce the size of the tuples. This implies that domain specific semijoins can be performed in a fragment-to-fragment manner and this in turn gives more flexibility.

The domain specific semijoin is defined as follows:

$$R_{ik}(A = B]R_{jm} = \{r \mid r \in R_{ik};\ r.A \in R_{jm}[B] \cup (Dom[R_j.B] - Dom[R_{jm}.B])\}$$

This is the approach used in horizontally fragmented databases. Tuples are horizontally partitioned into fragments such that each subset has some common properties. A relation is partitioned (fragmented) by attribute A if the domain of attribute A in the mth

fragment, Dom [$R_{im}$.A] and the domain of attribute A in the nth fragment, Dom [$R_{in}$.A] are disjoint for each pair of Rim and $R_{in}$ in $R_i$, that is $m \neq n$.

This means that suppose we are to join to fragments residing on different sites. One fragment is Rik and the other is Rjm of the relations Ri and Rj respectively. The joining attribute in Ri is A and in Rj is B. We take the attribute B in Rjm (fragment) and project Rjm over that attribute. There is a possibility that some tuples, which are in both the relations and are the contributive tuples would have been missed because the fragment is taken into account. Therefore, we union it with the Domain of B complement Domain of Rjm(to get the remaining) values of B. We then take this union to the site of Rik and see if there are any matching tuples. We keep the matching tuples at the site of Rik.

An interesting modification to this would be similar to that of matching and non-matching in the two-way semijoin. This implies instead of sending the union and thus a lot of intersite data transfer, we can send the non-matching tuples. In this case, we take the domain of B less the Rjm[B]. This would give us the non-matching tuples. We send these nonmatching values to the site where Rik resides and eliminate the tuples that are matching with the nonmatching values we have provided. This guarantees that we will always have minimum intersite data transfer.

## 2.5 APPROACHES

Algorithms for processing distributed queries require a priori estimates of the size of intermediate relations [BRP92]. Most such algorithms take a "static" approach in which the algorithm is completely determined before processing begins. If size estimates are found to be inaccurate at some intermediate stage, there is no opportunity to reschedule, and the result may be far from optimal. Adaptive query execution may be used to alleviate the problem. "In adaptive query optimization the query execution plan can be changed at any stage during query execution" p.253 [BRP92]. The terms dynamic and adaptive are used interchangeably throughout this thesis. Care is necessary, though, to ensure that the delay associated with re-scheduling does not exceed the time saved through the use of a more efficient strategy. This paper presents a low overhead delay

method to decide when to correct a strategy. Sampling is used to estimate the size of relations and alternative heuristic strategies prepared in a background mode are used to decide when to correct. Correction is made only if lower overall delay is achieved, including correction time. Evaluation using a model of a distributed database indicates that the heuristic strategies are near optimal. Moreover, it also suggests that it is usually correct to abort creation of an intermediate relation, which is much larger than predicted.

When the query is sent out, it is first parsed into its canonical form. The optimizer then formulates a query processing strategy QPS which specifies the sequence in which relational operations are executed and the network locations of their execution. The strategy is distributed to the relevant information processors, also known as cohorts, which participate in the strategy's execution. In static processing when a strategy is formulated it remains unchanged until it is executed.

Essentially we have to make sure that when we are formulating the strategy, we get a correct estimation of the size of the intermediate results. And static strategy assumes that all the estimates are accurate. Now we have two options:

1. Acquire accurate estimates.
2. Use other strategies.

Much work has been done to get or to obtain the estimates. However, they are often unobtainable. Moreover, in attempting to obtain those we have overheads in terms of size and maintenance of those estimates.

The second option is more feasible. Use of adaptive (dynamic) strategies is encouraged. What happens in this case is: Execution of a strategy is monitored and if at some intermediate stage a priori estimates used in its optimization prove to be inaccurate, corrections are made with updated information.

Once we decide to use the adaptive strategy we come up with an important question:

1. When to correct the strategy?

There are two ways to deal with it:

1. We reformulate the unexecuted portion of the strategy at every intermediate stage on the basis of available updated information. If the new strategy is estimated to reduce cost then the correction is appropriate. This is known as reformulation.
2. We have a threshold and if the intermediate results exceed this threshold then we will reformulate. This is known as the threshold method.

## 2.6 METHODS OF QUERY EXECUTION

**2.6.1 STATIC:** The strategy execution is static.

### 2.6.2 ADAPTIVE:

#### 2.6.2.1 ABORT JOIN OPTIMAL (AJO)

This method is theoretically optimal but unrealizable. It is similar to the proposed method but assumes perfect a priori knowledge of the intermediate result size just before the join execution commences. Without overhead delay, a new corrective strategy is formulated and executed for the remaining unprocessed portion of the strategy.

#### 2.6.2.2 COMPLETE JOIN OPTIMAL (CJO)

This is identical to the AJO method with the exception that the current join is not aborted. Only when the join is completed is a new strategy formulated and instituted without delay for the remaining unprocessed portion of the strategy.

#### 2.6.2.3 ABORTED JOIN LAST (AJL)

The proposed method proceeds as follows:

1. Give a query; a formulator/optimizer is used to derive a processing strategy. The strategy is distributed to cohorts, which then cooperate in transferring relations and executing relational operations according to the strategy's instructions.
2. Concurrently with the strategy's execution, an alternative strategy is formed for each intermediate result.
3. During the course of a join execution, a sampling method is used to estimate the size of the result. This estimate is then used to update the estimated delay of the current strategy and compare it with the delay of the alternative strategy. If the alternative strategy has a lower expected delay, correction takes place; the current strategy is aborted and the alternative strategy is adopted. Otherwise the original strategy is allowed to continue. If more than one processor decides to correct, consistent corrective strategy is adopted by all cohorts. The corrective strategy must be broadcast with a time stamp to all the cohorts to ensure the adoption of current corrective strategy.

Let $T_{\alpha,\beta}\{i, j, y\}$ be the delay of query i such that the size of the jth intermediate result of its strategy is increased by a factor y. Depending on the subscript $\alpha$, the strategy execution is assumed to be either static or adaptive. The subscript $\beta$, $0 < \beta \leq 1$, which applies to the proposed AJL method only, denotes the fraction of the join used by the sampling method to determine the size of the join result [BRP92].

$$\text{Therefore, } C_{\alpha,\beta}\{i, y\} = \sum_{j=1}^{M_i} T_{\alpha,\beta}\{i, j, y\} / M_i$$

$$D_{\alpha,\beta}\{y\} = \text{average delay over all N queries} = D_{\alpha,\beta}\{y\} = \sum_{i=1}^{N} T_{\alpha,\beta}\{i, j, y\} / N$$

A comparison of the idea of [BPR92] with the JAL dynamic algorithm is given in chapter 4.

# 3. KEY CONCEPTS

## 3.1 TRANSPARENCY: FRAGMENTATION AND REPLICATION

This is similar to the concept of encapsulation and data independence. Replication transparency is: If there are replicas of database objects, their existence should be controlled by the system and not by the user. This is because if the user starts moving the data then we will not be able to maintain the integrated and consistent data. However, if the user is aware of the existence of replicas and is responsible for their management, then the system has to do minimal work and the performance might be better.

If the database objects are fragmented, then the system has to handle the conversion of user queries defined on global relations to queries defined on fragments. The fragmentation transparency thus implies translation from global queries to fragment queries and putting together fragments into an answer.

## 3.2 DBMS IMPLEMENTATION ALTERNATIVES

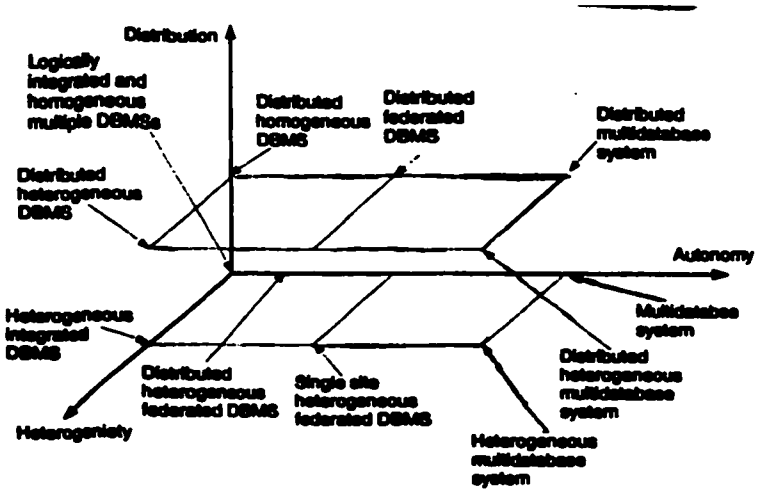



**Figure 3.1: DBMS Implementation Alternatives chapter 4, pp. 79 [OV91]**

The DDBMS has several alternatives for designing the database. They range from heterogeneous integrated database management to distributed, homogeneous multi-database management system as shown in the figure above. There are tradeoffs for each of designs on the basis of the three axes: distribution, heterogeneity and autonomy [BFM+00].

As for the correctness of the fragments, we have to ensure that:

1. The reconstructed relation from the fragment is complete.
2. There exists a relational operator to reconstruct the original relation.
3. The fragments are disjoint.

As stated earlier the fragments can be fully, partially or not replicated at all. The rule of the thumb is:

If, **read-only queries** **>= 1, then replication is advantageous.**
**Update queries**

Otherwise, it could cause problems.

### 3.3 STAGES IN QUERY PROCESSING

Essentially in a distributed database system, processing a distributed query involves data transmission among the different sites of the computer network. In a system where the communication cost is a dominant factor among the costs of system resources, that is, the local processing cost is negligible compared to the data transmission cost, it is essential to reduce the total amount of data transmission to process a distributed amount of data transmission to process a distributed query. In such a system, the optimal query processing strategy is defined as the strategy that can answer the query with minimum data transmission [Won77].

Given a distributed query in such a system, how to generate an optimal processing strategy has been of great interest to a number of researchers. Typically, a given distributed query is processed into the following three phases:

1. **Local Processing Phase**: This is at each site involved in the query processing, all local processing such as selections from relations and projections on the joining and target attributes is performed. In the case of a distributed database with replicated and or fragmented relations, we assume that a set of non-redundant copies referenced in the query is pre-selected before processing begins.

2. **Reduction Phase**: During this phase a semijoin program (a sequence of semijoins) is generated by the query optimization algorithm to reduce relations in a cost-effective way and thus, to reduce the total amount of data transmission.

3. **Final Query Processing Phase**: This selects a final processing site and sends all reduced relations to that site where the final query processing is performed to answer the query [Won77].

In this framework, the reduction phase is the crucial stage and the primary concern of a query optimization algorithm is to generate a semijoin program that will be used in this phase. The major difference from algorithm to algorithm lies in how to generate such semijoin programs.

## 3.4 ASSUMPTIONS, NOTATIONS AND DEFINITIONS

### 3.4.1 GENERAL ASSUMPTIONS

We will have various assumptions depending on the algorithm we are implementing. However, for most algorithms and including this thesis the following are the basic assumptions [BRP92]. Please note: The specific assumptions for our thesis work are given in chapter 4.

1. A distributed relational database management system with a number of independent nodes distributed geographically and connected via a point-to-point network.
2. Each node has local processing and storage capabilities.
3. The relations are distributed amongst the nodes and all nodes can access all the data.
4. Only select-project-join (SPJ) queries are considered.
5. A uniform and independent distribution of attribute values in assumed.

### 3.4.2 NOTATIONS

The following definitions explain the exact meaning of the word and the context it has been used in. Also, the notations used throughout this survey are consistent [KYY+83].

For a relation, $R_i$, where i = 1, 2, ..., m,

$n_i$: number of tuples,

$\alpha i$: number of attributes,

S ($R_i$): size of the relation in bytes,

$| R_i |$: the cardinality or the number of distinct rows/tuples of a relation, $R_i$.

For each attribute $d_{ij}$, the projection of relation i over attribute j, where, j = 1, 2,..., n, i of relation $R_i$,

$b_{ij}$: size in bytes of the data item in attribute $d_{ij}$,

$| d_{ij} |$ is the cardinality of distinct attribute values of the projection of relation i over attribute j.

$S(d_{ij})$: is the size of the projection in bytes.

$|D(d_{ij})|$: is the cardinality of the domain for attribute $d_{ij}$. It is the number of possible values in the domain, not a count of the actual values occurring in the database. We assume that the cardinality is finite and known.

$p(d_{ij})$: is the selectivity of attribute $d_{ij}$. It is defined as the ratio of the number of different values occurring in the attribute over the total number of possible values of the attribute as stated in the following mathematical form:

$$|d_{ij}| / |D(d_{ij})|$$

Suppose, we have a relation such that $d_{ij}$, the cardinality of the projected tuples, is 200 and the total domain size is 1000. The $p_{ij}$ is as follows:

200/1000 = 0.2.

This means that the reducing power of the attribute $d_{ij}$ is 0.2. This figure is important because when we perform the joins we can use this figure to find out to what extent the attribute $d_{ij}$ can reduce another relation.

This implies that if we use $d_{ij}$ to join with another relation $R_k$ over attribute j such that $R_{kj}$ exists, we can use the selectivity figure to find out the reduction. Assuming $R_k$ has the size of 4000, the new $R_k'$ will have the size of:

$$S(R_k') = pd_{ij} * S(R_k)$$

$= 0.2 * 4000$

$= 800$

The **cost** of performing the semijoin is given as follows:

$$C(d_{ij} \rtimes d_{kj}) = C0 + S(d_{ij}) * C1$$

Sometimes for simplicity we assume that C0 is zero and C1 to be one.

### 3.4.3 FORMAL DEFINITIONS

1. **Select**: It filters or selects a subset of a relation using a given select condition that is a Boolean expression on the attributes on input relation. The result is the relation of input tuples satisfying the selection condition. As shown below:

$$\sigma_F(r) := \{t \mid t \in r \wedge F(t) = \text{true}\} \text{ for } r(R)$$

The resulting relation has the same attributes as the input relation.

2. **Project**: Selects or filters columns (attributes) from a relation. Attribute list is the list of attribute names from input relation [ME92]. Result is the relation consisting of projected attributes. Note the duplicate tuples are removed:

$$\Pi_X(r) := \{t(X) \; t \in r\} \text{ for } r(R)$$

3. **Join**: Join is Cartesian Product following a selection. Join condition is a Boolean expression on the attributes on input relations. The join on attribute A is denoted by $(R_i.A \bowtie R_j.A)$, where we are joining $R_i$ and $R_j$. The join is obtained by concatenating each row of $R_i$ with each row of $R_j$ whenever the values of $R_i.A$ and $R_j.A$ are equal. This is also known as equijoin. When the attributes in $R_i$ and $R_j$ have the same attribute name, this join is called natural join (another kind of equijoin) as this occurs naturally. For our survey we will be using natural join [ME92].

4. **Semijoin**: If A is a common attribute, then the semijoin over A is denoted by $R_j\text{-}A\rightarrow R_i$, where $R_j$ is the sending relation, $R_i$ is reduced relation and A is the joining attribute. $R_i \rtimes R_j$ also denotes it. This can be interpreted as projecting $R_j$ over attribute A to get the projection $R_j[A]$, shipping the projection $R_j[A]$ to the site of relation $R_i$, and then executing $R_j[A] \ltimes R_i$ [MB96].

Two-way semijoin is the two-way semijoin of relation $R_j$ by relation $R_i$ over attribute A obtained by performing two semijoins. The first being the semijoin of relation $R_j$ by $R_i$ and the second being the semijoin of relation $R_i$ by relation $R_j$[Won77]. It can be said that a two-way semijoin is obtained adding backward reduction to the semijoin.

## 3.5 QUERY GRAPH

Depending on a query sometimes a relation in the query might not be fully reduced. This implies the total intersite data transfer is still very high. Therefore, there emerges a need to characterize the queries whose referenced relations were fully reduced in order to differentiate them from those which were not completely reduced [YC84]. This characterization is facilitated by query graph, G = (V, E) where,

G is the query graph;

V is the set of vertices; and,

E is the set of edges.

It is simply an execution graph to represent the semijoin programs associated with the distributed processing of the query. The vertices of the query graph including the root node are the relations appearing in the qualification. An edge between nodes $R_i$ and $R_k$ exists if $Ri.Aj = R_k.A_j$, where $A_j$ is the joining attribute. If there are two joining attributes then the edge is labeled $\{A_j, A_h\}$.

The following is an example of a query graph:

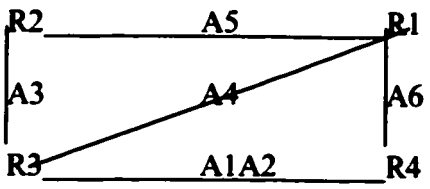



**Figure 3.2: Query Graph**

The above graph shows that R1 has a common attribute A5 with R2 and A4 with R3. R2 has A3 with R3 and R3 and R4 share A1 and A4. R4 share A6 with R1.

The query graph can be used both to represent the parallel execution and the serial execution. However, the way we represent the query graph, also known as rearrangement technique, can help us improve the associated semijoin program.

The following are the four main groups of query strategies:

### 3.5.1 SIMPLE QUERIES

Generally in a distributed database, it is better to do local processing first, because it reduces the amount of data to be transmitted.

Initial local processing results in the following parameters:

m: number of relations in the remaining query

$\alpha_i$ : number of attributes in relation $R_i$ ,

$\beta_i$ : number of internodal joining attributes in relation $R_i$ .

After initial local processing, queries become simple queries, the relations contain only one attribute: the joining attribute.

So, $\alpha_i = \beta_i = 1$ for i = 1, 2, ... , m.

The vital properties of simple queries are:

1. All relations should appear on directed path. This implies that an optimal strategy for a simple query is a string of directed edges, Ri1→Ri2→Ri3→...→Rit [YC84].
2. All relations should appear in increasing order of size [HY79].

Therefore, the optimal strategy is R1→R2→R3→...→Rt, where |R1| < |R2| < |R3| <... <|Rt|.

Hevner and Yao used this in the algorithm General introduced [AHY83].

### 3.5.2 QUERY TREE

A query tree (QT) is a finite tree in which [BC81]:

1. leaves represent the relations to be joined;
2. each leaf has associated the list of sites where its relation is available for materialization;
3. internal nodes as well as the root represent the join operators; and,
4. the root has associated a label representing the query source site, that is, the site where the result relation has to be stored and displayed.

A binary query tree BQT is a QT where non-leaf nodes have exactly two sons. If the query graph of a query is connected then the query equivalent to Q will also have a connected query graph. A query, Q, is called a tree query if either itself or an equivalent query has a tree query graph, otherwise it is a cyclic query. In a cyclic query none of the query graphs of equivalent qualification is a tree. If the query is a tree query, the relations of a query can be fully reduced by semijoins. On the other hand, semijoins may be inadequate to reduce a cyclic query. However, a cyclic query can be transformed into a tree query. There are different algorithms such as:

1. A relation merging algorithm [GS82] [KYY+83]
2. A tuple wise decomposition algorithm
3. An attribute addition algorithm [ER82]

### 3.5.3 CHAIN QUERY

A chain query is defined as a query having its graph configured as a chain with a target relation at the end. Chiu, Bernstein and Ho suggested an optimal algorithm for processing chain queries [CBH84]. The chain query intuitively joins each pair of adjacent relations, e.g., Ri and Ri+1 and then projects onto the attributes of R1 as shown below:

$$(R1 \bowtie R2 \bowtie R3 \bowtie \ldots \bowtie Rn)[R1]$$

where,

$Ri \cap Rj \neq \Phi$ if, $j=i-1$ or $j=i+1$

$Ri \cap Rj \neq \Phi$ if, $j<i-1$ or $j>i+1$

### 3.5.4 STAR QUERIES

As the name implies it is a query whose graph is configured as a star with the target relation in the center. It is usually in the networks with star topologies. Star queries are a special subset of tree queries. INGRES system at the University of California Berkeley was the first to introduce a general form of star queries expressed in QUEL [MSW75].

The following is an example of the star query graph and the query language:

**RETRIEVE** R0.A1, R0.A2...R0.Am)

**WHERE** R0.X1= R2.X1 And ... And Rx.Xn=Rn.Xn

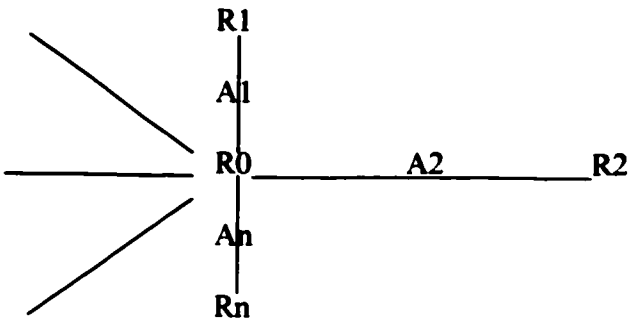



**Figure 3.3: Star Query Graph**

# 4. JAL: THE DYNAMIC APPROACH

This chapter describes in depth the JAL Dynamic algorithm we have proposed, along with the bench mark heuristic: AHY algorithm [AHY83]. It is important to remember that we are concerned with making query optimization adaptive and therefore, as stated in chapter 2, the terms adaptive and dynamic are used interchangeably.

AHY algorithm is considered as the best by many researchers who are working on distributed query optimization. Therefore, it is used as a yardstick in terms of measuring the performance of the JAL dynamic algorithm.

In the first section the assumptions are stated as mentioned by [AHY83] and the same assumptions are retained in our JAL dynamic algorithm. The next section states the thesis problem. A detailed description of the AHY algorithm is given. The explanation is illustrated with a worked example to make it clearer. The proposed JAL dynamic algorithm follows this. The same example is used to give a comparison between the two approaches.

## 4.1 THESIS ASSUMPTIONS

The following are the assumptions made by AHY algorithm in [AHY83] and these exact assumptions hold true for the JAL dynamic algorithm:

1. The database is viewed logically in the relational data model. The following statistics are available in a file (the values are either integer or double):

For each relation, $R_i$, i = 1, 2, ..., m,

$n_i$: number of tuples,

αi: number of attributes,

si: size (e.g., in bytes).

For each attribute dij, j = 1, 2, ..., αi for relation Ri,

pij: selectivity,

bij: size (e.g., in bytes) of the data item in attribute dij.

2. Simple queries will be run and the database will be homogeneous.

3. To process a query in a relational database, only restriction, projection and join will be used.
4. The query processing is run on a dedicated system to achieve execution times. Dynamic system factors such as communication line contention and subsequent queuing delays are not considered.
5. The response time is considered to be equal to the units of data transferred. Therefore, if 1 byte of data is being transferred the time is 1unit.
6. The start-up cost of 20 units of time is added to each processing stage.
7. The following file (also known as query) is used as the input file in the experiments:

```
3 2
 1000  400 .62 100 .7
 2000  400 .786  450 .9
 3000  900 .81 600  .83
```

**INPUT FILE FORMAT**

The first line:

Number of joining relations; space; number of joining attributes.

All the subsequent lines:

Size of the relation; space; size of the first joining attribute; space; its selectivity; space; the size of the second joining attribute; space; its selectivity.

## 4.2 THESIS PROBLEM

Given the input file (format given in Section 4.1 Assumption 7), our objective was to implement the AHY algorithm and the JAL dynamic algorithm and compare the time difference of query response time.

## 4.3 AHY ALGORITHM – A DETAILED LOOK

In 1983 Apers, Hevner and Yao presented an algorithm in their paper, "Optimization Algorithm For Distributed Queries" commonly known as Algorithm General or AHY algorithm [AHY83]. This algorithm was a modification of previous algorithms known as

parallel and serial explained in Chapter 2. Since the objective of this thesis is to minimize the response time, we only focused on the parallel version of the AHY algorithm.

The algorithm creates the candidate schedules for each joining attribute. The data transmissions used for reducing a relation and the transmission of the reduced relation to the query computer form a **schedule** for this relation [AHY83].

Suppose we have three relations – R1, R2 and R3. We are treating each joining attribute as if it were a simple query. Suppose the joining attribute is j and therefore, we have the sizes of j (the joining attribute) for each Ri to be d1j, d2j, and d3j. We sort the dijs and assuming they are sorted, we create the candidate schedules as follows:

1. For the smallest dij, which is d1j, the d1j is the candidate schedule.
2. For the second we take the smaller of either d2j or the concatenation of d1j with d2j.
3. For all the rest of the relations we send all the dijs in parallel to the unknown site right now.

Once it has created the candidate schedules for each joining attribute, we take a relation at a time and check which joining attributes can reduce a particular relation. We choose the minimum response time for each relation. The final response time for the query is the maximum of all the response times we have.

### 4.3.1 PSEUDO CODE FOR THE AHY ALGORITHM

The concise description of AHY algorithm is given below [AHY83]:

1. Do all initial local processing.
2. Generate candidate relation schedules. Isolate each of the joining attributes and consider each to define a simple query with an undefined node.
   a. Order the relation Ri so that $S_1 <= S_2 <= \ldots <= S_m$ (Si: Size of $R_i$)
   b. Consider each relation Ri in ascending order of size.

c. For each relation $R_j$ (j<i), construct a schedule to $R_i$ that consists of the parallel transmission of the relation $R_i$ and all schedules of relations $R_k$ (k < j). Select the schedule with minimum response time.

3. Integrate the candidate schedules. For each relation Ri, the candidate schedules are integrated to form a processing schedule for Ri. The integration is done by the procedure called Response, identical to above except that it is applied to relations rather than attributes.
4. Remove schedule redundancies. Eliminate schedules for relations which have been transmitted in the schedule of another relation.

### 4.3.2 A WORKED EXAMPLE

The following example clarifies the AHY algorithm as it is explained stepwise.

| Relation (Ri) | Size (S(Ri)) | di1 | | di2 | |
|---|---|---|---|---|---|
| | | Size of (di1) | Selectivity (pi1) | Size of (di2) | Selectivity (pi2) |
| R1 | 1000 | 300 | 0.9 | 300 | 0.25 |
| R2 | 2000 | 100 | 0.9 | 900 | 0.75 |
| R3 | 2000 | 400 | 0.8 | 600 | 0.5 |

**Table 4.1: The statistics of the relations used in the distributed Query Optimization**

The above table gives the statistics of the relations which is used in the algorithm. There are three relations R1, R2 and R3. The columns give the sizes of each relation; the sizes of the joining attributes and their selectivities.

**A. Generate the Candidate schedule for each joining attribute:**

We assume the initial start up cost to be 20.

1. We take each joining attribute or consider relations as simple queries and sort them according to the attribute size. Therefore, we have: d21 < d11 < d31.
2. Generate candidate schedules (CSCHL) [AHY83] for each of these attributes.

**Candidate Schedules for Attribute_1:**

CSCHL for d21 is:

(d21)+-----120.00-----+

The response time is calculated as: 100 (Size of the attribute) + 20 (Start-up cost) = 120.

**Best Response time is: 120.**

CSCHL for d11: We first take the d21 and serial it with d11. This gives us one response time. The second way of getting the response time is sending d11 by itself. The **best response time** is the minimum response time of the two which in this case **is: 320.**

Sending d11 by itself:

(d11)+-----320.00-----+ (I)

Or,

(d21)+-----120.00-----+(d11)+-----290.00-----+ (II)

The cost of (II) is calculated as follows:

(p21 * d11) + 20 + response time of d21.

= (.9 * 300) + 20 +120

= 410.

CSCHL for d31:

For all the other joining attributes we send everything in parallel. Therefore the CSCHL for d31 is:

(d21)+-----120.00-----+

|

+(d31)-----344.00-----+

|

(d11)+-----320.00-----+ **Best Response time is: 664.**

Again we calculate the time 164 as follows:

(p21 * p11 * d31) + 20 + response time of d11

= .9 * .9 * 400 + 20 + 320

= 344.

Therefore, Candidate schedules in ascending order:

d21 's candidate schedule = 120.00

d11 's candidate schedule = 320.00

d31 's candidate schedule = 664.00

We do the same thing for the other joining attribute.

Candidate Schedules for Attribute_2: We start with d12, since d12 < d32 < d22

CSCHL for d12:

(d12)+-----320.00-----+ **Best Response time is: 320.**

CSCHL for d32:

(d12)+-----320.00-----+(d32)+-----170.00-----+

This gives a response time of: 490.

Or,

(d32)+-----620.00-----+

We choose the minimum therefore, **the best response time is: 490.**

CSCHL for d22:

d22 = 622.50:

(d12)+----------320.00----------+
|
+(d22)-----132.50-----+
|
(d12)+---320.00---+(d32)+---170.00----+

**The best response time is: 622.50.**

**Therefore,** Candidate schedules in ascending order:
d12 's candidate schedule = 320.00
d32 's candidate schedule = 490.00
d22 's candidate schedule = 622.50.

**B. Once we have created the CSCHL then we take each relation one at a time and identify the joining attributes that can reduce it. We take the CSCHL for those attributes and sort them and construct an integrated schedule using the same algorithm we used to generate the CSCHL.**

**REDUCING R1:**
R1 can be reduced by the following CSCHLs in ascending order:
d21: 120.00 d32: 490.00 d22: 622.50 d31: 664.00
Therefore, we integrate these candidate schedules using their response/arrival time (RT).
Please note that if an attribute is being used twice then selectivity is used only once.

**RT_21: 1040.00**

(d21)+-----120.00-----+(R1)-----920.00-----+

Response time is calculated as follows:

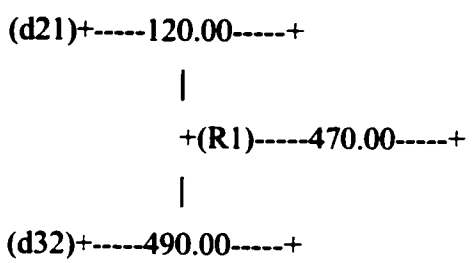
p21 * S(R1) + 20

= .9 * 1000 + 20

= 920.00.

For all the rest of the reducing attributes send everything in parallel:

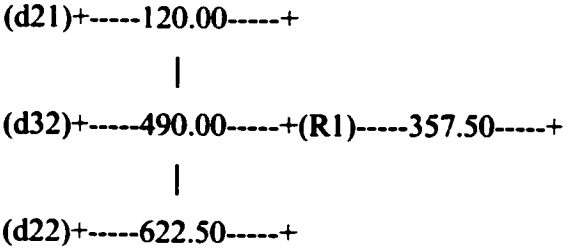
**RT_32: 960.00**



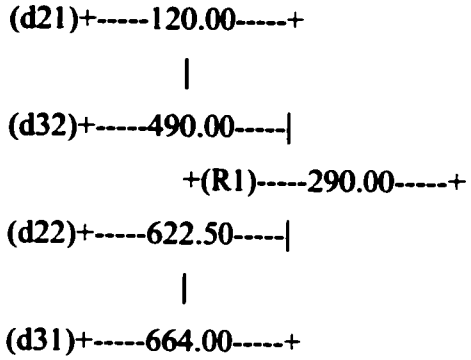

The response time is calculated as:

p21 * p32 * S(R1) + 20 + response time for the d32.

**RT_22: 980.00**



**RT_31: 954.00**

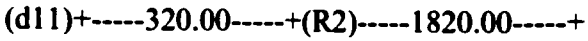



Once we have calculated all the response times for reducing R1, the best response time is the minimum of these. Therefore, **R_1(min)= 954.00.**

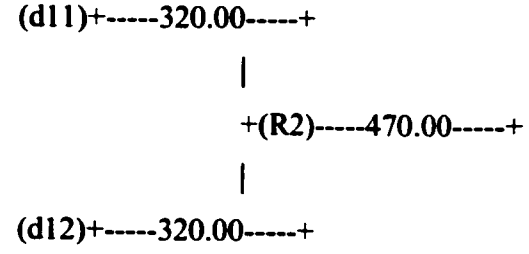
**REDUCING R2:**

For R2, CSCHLs in ascending order:
d11: 320.00 d12: 320.00 d32: 490.00 d31: 664.00

**RT_11: 2140.00**




**RT_12: 790.00**



**RT_32: 735.00**

(d11)+-----320.00-----+
|
(d12)+-----320.00-----+(R2)-----245.00-----+
|
(d32)+-----490.00-----+

**RT_31: 864.00**

(d11)+-----320.00-----+
|
(d12)+-----320.00-----|
+(R2)-----200.00-----+
(d32)+-----490.00-----|
|
(d31)+-----664.00-----+

**R_2(min)= 735.00.**

**REDUCING R3:**

CSCHLs for R3 in ascending order:

d21: 120.00 d11: 320.00 d12: 320.00 d22: 622.50

**RT_21: 1940.00**

(d21)+-----120.00-----+(R3)-----1820.00-----+

**RT_11: 1960.00**

(d21)+-----120.00-----+

|

+(R3)-----1640.00-----+

|

(d11)+-----320.00-----+

**RT_12: 745.00**

(d21)+-----120.00-----+

|

(d11)+-----320.00-----+(R3)-----425.00-----+

|

(d12)+-----320.00-----+

**RT_22: 946.25**

(d21)+-----120.00-----+

|

(d11)+-----320.00-----|

+(R3)-----323.75-----+

(d12)+-----320.00-----|

|

(d22)+-----622.50-----+

**R_3(min)= 745.00.**

Once we have calculated the minimum response times for each relation, we take the maximum of those times to be the actual time for the query response. **The final query response time in this case is: 954.00.**

## 4.4 THE JAL DYNAMIC ALGORITHM

AHY algorithm provides a very efficient query plan for simple queries. However, one important drawback of the AHY algorithm is that it is static in nature. This implies that once a query plan has been made, there is no way we can change the plan. It strictly follows a given pattern and provides the response based on the query plan. The JAL dynamic algorithm is an enhancement to the existing algorithm.

We are proposing a dynamic algorithm which leads to dynamic query plans and thus, efficient execution of the query. The JAL dynamic algorithm differs from the AHY algorithm in two ways:

1. The JAL dynamic algorithm does this by calculating the profit of a semijoin to take place. If **the profit is > 0**, then that semijoin is allowed to take place; otherwise continue with the other semijoins.

When a semijoin is performed, there are some overheads associated with it. The **cost** is the amount of data transmitted from the site of one relation to the other. One important point to remember is the fact that we are not concerned with the cost of the first joining attribute to be sent. This is because we have to start at some point. This is in line with the AHY algorithm. Following that all the costs are taken into account. Therefore, the cost of d21 (provided that d11 was sent before it) is:
p11 * d 21 ( the selectivity of p11 times the size of d21). Moreover, we take the combined selectivity values for any further join as the cost increases with the number of given joins being performed. Therefore, if the sequence is d11, d21, d31, then the cost is: p11 * p21 * d31.

The **benefit** is the actual amount of data we do not require in the final result, in turn saving our costs. The following formulae explain the cost, benefit and profit calculations:
Let,
x = Attribute size [i];

y = Attribute size [i-1];

z = Combined Selectivity;

Cost = x * z (the cumulative/combined selectivity times the attribute size of the current joining attribute);

Benefit = (1-z) * y (the cumulative/combined selectivity saved as that amount of data was not transmitted times the previous attribute size).

Profit = Benefit – Cost.

2. The JAL dynamic algorithm also improves the CSCHL calculation step by introducing a threshold. This threshold was calculated by trial and error method. The value for this threshold is .2. This threshold is used with a condition, and it not only improves the response time but it also gets rid of the redundancies in the AHY algorithm.

### 4.4.1 PSEUDO CODE FOR THE JAL DYNAMIC ALGORITHM

1. For each joining attribute:
    1. For the first Candidate schedule send the joining attribute by itself.
    2. For the second Candidate schedule check:

    If (pkj – pij > .2), serial pkj and pij (dij < dkj);

    Else, send pkj by itself.

    3. For all candidate schedules after this calculate profit:

    If (Profit > 0), generate the candidate schedule;

    Else, continue.
2. For each Relation:
    1. Check which candidate schedules can reduce the relation, generate the minimum response time for each relation.
3. The maximum of the minimum times is the query response time.

### 4.4.2 A WORKED EXAMPLE

We use the same table to compare the JAL dynamic algorithm with the AHY algorithm.

| Relation (Ri) | Size (S(Ri)) | di1 | | di2 | |
|---|---|---|---|---|---|
| | | Size of (di1) | Selectivity (pi1) | Size of (di2) | Selectivity (pi2) |
| R1 | 1000 | 300 | 0.9 | 300 | 0.25 |
| R2 | 2000 | 100 | 0.9 | 900 | 0.75 |
| R3 | 2000 | 400 | 0.8 | 600 | 0.5 |

**Table 4.2: The statistics of the relations used in the distributed Query Optimization**

**A. Generating the CSCHL for each joining attribute:**

**Candidate Schedule for Attribute 1:**

CSCHL for d21:

d21 = 120.00:

(d21)+-----120.00-----+

d11 = 320.00:

(d11)+-----320.00-----+

Candidate schedules in ascending order:

d21 's candidate schedule = 120.00

d11 's candidate schedule = 320.00

d31 's candidate schedule = 0.00.

Note, by using the threshold .2, we eliminated the need to compare and get the minimum at the second step. The profit calculation also helped us remove the need to get the CSCHL for d31.

**Candidate Schedules for Attribute 2:**

d12 = 320.00:

(d12)+-----320.00-----+

d32 = 490.00:

(d12)+-----320.00-----+(d32)+-----170.00-----+

d22 = 622.50

(d12)+------------320.00------------+
|
+(d22)-----132.50-----+
|
(d12)+---320.00---+(d32)+---170.00----+

Candidate schedules in ascending order:

d12 's candidate schedule = 320.00

d32 's candidate schedule = 490.00

d22 's candidate schedule = 622.50

**B. For each relation, find which candidate schedules can reduce it and use the appropriate CSCHL calculated in step A. to reduce them.**

**REDUCING R1:**

The CSCHL for R1 in ascending order:

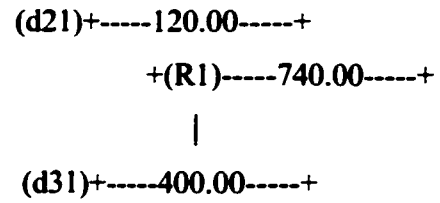
Please note that we did not calculate the CSCHL for d31 as it was not profitable. Therefore, we do not use its candidate schedule but in fact we send it directly to reduce R1.

d21: 120.00 d32: 490.00 d22: 622.50

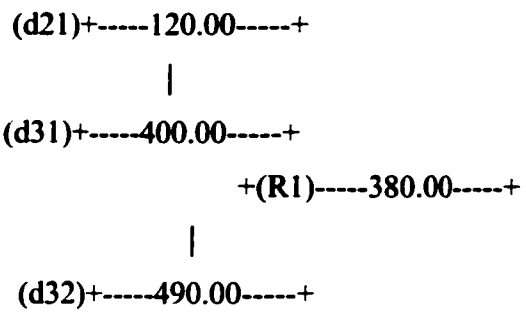
RT_21: 1040

+(R1)-----920.00-----+
|
(d21)+-----120.00-----+

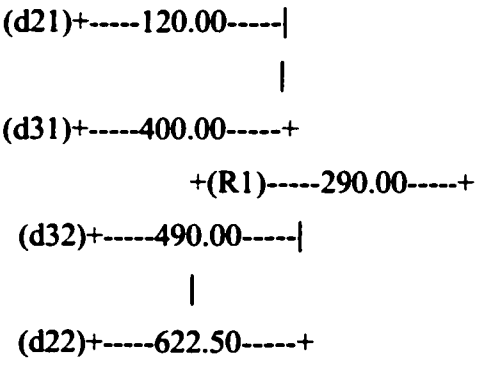
RT_31: 1140.00




RT_32: 870.00




RT_22: 892.50




**R1(min)= 870.00**

**REDUCING R2:**

Note, we are again not considering d31 as it was not a profitable join:

d11: 320.00  d12: 320.00  d32: 490.00

RT_11: 2140.00

+(R2)-----1820.00-----+

|

(d11)+-----320.00-----+

RT_12: 790.00

(d11)+-----320.00-----+(R2)-----470.00-----+

|

(d12)+-----320.00-----+

RT_31: 780.00

(d11)+-----320.00-----+

|

(d12)+-----320.00-----+

+(R2)-----380.00-----+

|

(d31)+-----400.00-----+

RT_32: 690.00

(d11)+-----320.00-----|

+(R2)-----200.00-----+

(d12)+-----320.00-----|

|

(d31)+-----400.00-----+

|

(d32)+-----490.00-----+

**R2(min)= 690.00**

**REDUCING R3:**

R3 can be reduced by the CSCHLs:

d21: 120.00  d11: 320.00  d12: 320.00  d22: 622.50

RT_21: 1940.00

(d21)+-----120.00-----+(R3)-----1820.00-----+

RT_11: 1960.00

(d21)+-----120.00-----+
|
+(R3)-----1640.00-----+
|
(d11)+-----320.00-----+

RT_12: 745.00

(d21)+-----120.00-----+
|
(d11)+-----320.00-----+(R3)-----425.00-----+
|
(d12)+-----320.00-----+

RT_22: 946.25

(d21)+-----120.00-----+
|
(d11)+-----320.00-----|
+(R3)-----323.75-----+
(d12)+-----320.00-----|
|

(d22)+-----622.50-----+

**R3(min) = 745.00**

The final response time is, of course, the maximum of the response times for reducing each relation. It is 870.00 in this case.

## 4.5 COMPARISON OF JAL DYNAMIC ALGORITHM WITH [BPR92]

In 1992, P. Bodorik, J. Riordan, and J. Pyra [BRP92] introduced the idea for making query optimization dynamic. [PBR92] uses two approaches in order to make their algorithm dynamic:

1. Threshold method: In the process of strategy formulation, additional information is also supplied to support the decision-making. Two threshold values: Vlow and Vhigh are used. A strategy is corrected if the actual values are not within this range. This method makes use of "Critical Path Network" if the partial results constructed are critical to the query response.
2. Reformulation: As the new partial results are formed, the unexecuted portion of the query is reformulated using the most up to date information available. Correction is encouraged if the new reformulation has a lower cost than the cost of the current strategy.

To determine if a strategy is proceeding as planned, information regarding the progress of the strategy is to be gathered by one or more processors.

This is called "monitoring." Notwithstanding the monitoring method used, correction is desirable based on the two-aforestated ways.

The JAL dynamic algorithm based on the idea of making query optimization dynamic is very different from these values. This is because there is no kind of "monitoring" involved, or there is no point in time when the corrections are made based on partial results or threshold values. Although, the JAL dynamic algorithm makes use of the

threshold value of .2, that threshold is not a range for the partial results, monitored and then corrected if need be. In fact, this threshold just acts as a condition, thereby reducing the time and expediting the execution of the process.

## 4.6 COMPARISON OF JAL DYNAMIC ALGORITHM WITH [Bea95]

[Bea95] also worked on making the AHY algorithm dynamic but the AJL dynamic algorithm is proved to be better because it takes into account the following important points:

1. [Bea95] was focusing on reducing total time or the total data transfer in contrast to our work, which is focusing on the Parallel version of the AHY algorithm: to reduce the query response time.
2. [Bea95] took just the selectivity value for a given joining attribute and times it with the attribute size. Our algorithm takes the combined selectivity, which ensures that all the costs are taken into account.
3. [Bea95] while calculating the benefit, multiplies the selectivity value with the size of the relation. However, when we are generating the candidate schedules at that time we are not dealing with the relations, but the joining attribute and, therefore, we are multiplying it by the size of the joining attribute. This is a realistic way of calculating the benefit.

## 4.7 COMPLEXITY ANALYSIS OF THE JAL DYNAMIC ALGORITHM

### 4.7.1 COMPLEXITY IN TERMS OF MEMORY USAGE

The algorithm in its implementation is making use of 3 main arrays in order to store the input file, candidate schedules and the relations. This helps us make efficient use of memory and ensures that no space is wasted. At the same time, it is important to note that the prime concern of this algorithm is response time and not memory usage. However, in our implementation we have ensured that make use of memory efficiently by making use of arrays.

### 4.7.2 TIME COMPLEXITY

The algorithm can be divided into two main stages:

Sorting of joining attribute for each relation:

In our implementation we ensure that an efficient sorting algorithm is used in order to maintain consistency and give results in minimum time. Quick sort is used; it has the time complexity of O(mlogm), where m is the number of relations.

Creating candidate schedules and reducing relations:

If a general query is sent which requires m relations and there are $\Phi$ joining attributes, then in the first step of AHY algorithm when we are adding the candidate schedules, there are no more than $O(\Phi m)$ candidate schedules which have to be added.

For each relation, we have to determine the best (minimum) response time among the $O(\Phi m)$ candidate schedules. Therefore, the complexity of this step is $O(\Phi m^2)$. Hence, the overall complexity of the JAL dynamic algorithm is $O(\Phi m^2)$.

# 5. TESTING, EXPERIMENTATION AND EVALUATION

Query optimization is a NP-hard problem [AHY83]. Therefore, most of the work is actually analytical comparisons made among algorithms. These empirical studies provide the best measure of validating the techniques proposed.

## 5.1 METHODOLOGY

It is important to understand that our evaluation techniques and most of the previous heuristics are made on Select-Project-Join (SPJ) queries. Although, it might look like a constraint, but in reality we are not restricting the type of query being tested. This is because it is possible to translate any query to SPJ query and for most of the work simple query is considered to evaluate the exact response time required in joining two relations.

1. The experiments compared and evaluated the AHY algorithm and the JAL dynamic algorithm.
2. The performance of both the algorithms was analysed on the basis of simple queries.

### 5.1.1 THE TEST QUERIES

We assume that the query is being evaluated after all local processing has been done. This is in line with our methodology of testing the simple queries. We do not consider any cost incurred during local intra site processing. Moreover, to maintain consistency we add an arbitrary STARTUP cost of 20 to all the joins being performed.

Various parameters were altered to construct queries with following characteristics:

1. The query was run starting 3 and up to 6 relations and number of joining attributes was be varied between 2 and 4. We can have 2 relations with 3 joining attributes, etc. Therefore, we can have up to a maximum of 12 different types of test queries.
2. Each relation has tuples between 800 and 6000 tuples.

3. The cardinality of the domain of the joining attributes ranges from 500 – 1500.
4. In order to come up with concrete and realistic results, we considered different levels of selectivities. We tested:
   a. 1000 queries with low selectivities ($p <= .33$) of the joining attributes.
   b. 1000 queries with medium selectivities ($.34 < p <= .66$) of the joining attributes.
   c. 1000 queries with high selectivities ($.67 < p <= .99$) of the joining attributes.

### 4.1.2 CREATION OF STATISTICS

The statistics used by both AHY and the JAL dynamic algorithm is created as follows:

1. Firstly, run the create_query.c program file to create the "dbstats file." This program requires two command line arguments: Number of relations and the number of joining attributes. The dbstats file with selectivities is generated.
2. Once, we have dbstats file, we can use relbuilder.c program, which takes the dbstats file and creates the actual relations based on the dbstats files. It also creates the domains of the joining attributes.
3. Finally use the dbstats file to run the AHY and the JAL dynamic algorithms to compare the response times for a given query.

### 5.1.3 THE TEST DATABASE

The following are some advantages for using the statistical representation of the relations to test our queries:

1. For the execution of the queries there is no need to construct the entire database, only the relations required are constructed.
2. The actual values of the key attributes are randomly selected from the pool of the values in the domain.
3. This database helps us adhere to the format of input file used by [AHY83].

## 5.2 EXPERIMENTAL RESULTS

We created the dbstats file randomly and used it as an input for both the AHY and the JAL dynamic algorithms. The implementation was done in C language and the program was run on the davinci server with a RAM of 8192 MB; 40 hard disks (4 with 18 GB and 32 with 36GB) and 12 CPU each with 336 MHz on Sun Ultra Sparc-II with RAM 768 MB and CPU speed of 450 MHz. The operating system was Unix 5.

The following tables have 4 columns. The first column is the query type. It specifies the number of relations and the joining attributes. For each selectivity value, 1000 queries were run, therefore, on average 85 queries were run for each query type. The next two columns the average response time for each query type are stated. The final column gives the percentage difference between the AHY and the JAL dynamic algorithm.

| Type | AHY | JAL | Percentage Improvement |
|---|---|---|---|
| 3-2 | 388.56 | 388.56 | 0 |
| 3-3 | 434.28 | 434.28 | 0 |
| 3-4 | 451.67 | 451.67 | 0 |
| 4-2 | 468.69 | 468.69 | 0 |
| 4-3 | 487.54 | 487.54 | 0 |
| 4-4 | 499.01 | 499.01 | 0 |
| 5-2 | 542.32 | 542.32 | 0 |
| 5-3 | 559.69 | 559.69 | 0 |
| 5-4 | 589.29 | 589.29 | 0 |
| 6-2 | 597.58 | 597.58 | 0 |
| 6-3 | 611.21 | 611.21 | 0 |
| 6-4 | 659.59 | 659.59 | 0 |
| **Average** | 524.11 | 524.11 | 0 |

**Table 5.1: The percentage improvement in the response time with selectivity value < = .33**

The above table shows the percentage time difference for the 12 different query types with the selectivity values < = .33.

It was found that the JAL dynamic algorithm gives equal response time when the selectivity value is < = .33.

The following graph illustrates the values from table 5.1:

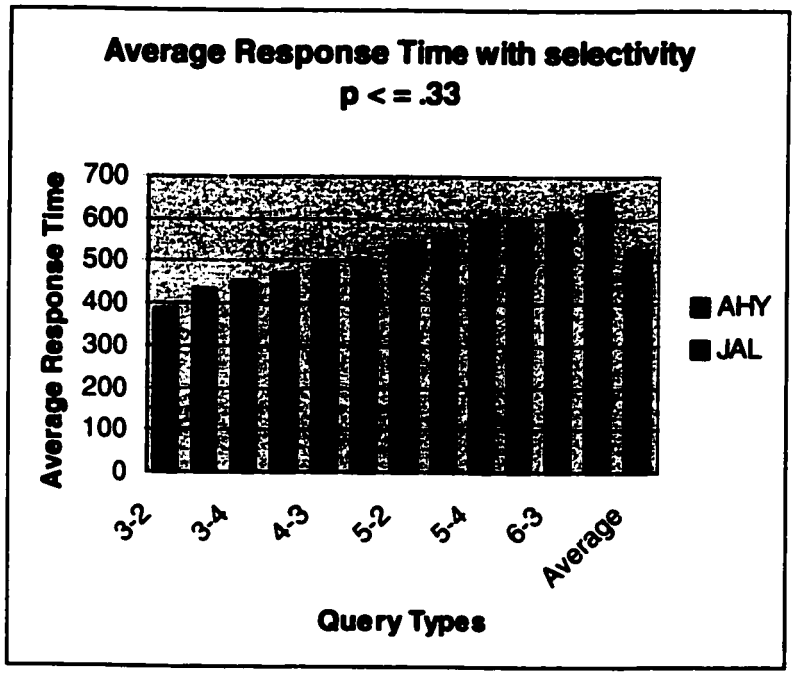



**Figure 5.1: Average response time with selectivity value p < = .33**

The query response time for both the AHY and the JAL dynamic algorithm are the same when the selectivity values are < = .33.

| Type | AHY | JAL | Percentage Improvement |
|---|---|---|---|
| 3-2 | 803.95 | 659.00 | 19 |
| 3-3 | 725.77 | 685.67 | 5.1 |
| 3-4 | 750.89 | 692.64 | 8.41 |
| 4-2 | 910.88 | 729.41 | 24.88 |
| 4-3 | 875.88 | 776.56 | 12.79 |
| 4-4 | 882.71 | 801.52 | 10.13 |
| 5-2 | 889.71 | 829.65 | 7.24 |
| 5-3 | 995.17 | 850.36 | 17.03 |
| 5-4 | 1066.18 | 936.73 | 13.82 |
| 6-2 | 1106.94 | 948.29 | 16.71 |
| 6-3 | 1078.25 | 975.97 | 10.48 |
| 6-4 | 1117.25 | 1019.48 | 9.59 |
| **Average** | 932.66 | 825.44 | 12.99 |

**Table 5.2: The percentage improvement in the response time with selectivity value .34 < p < = .66**

The above table shows the percentage time difference for the 12 different queries with the selectivity values .34 < p < = .66.

It was found that when the selectivity values are in the range of .34 - .66, in most of the cases the JAL dynamic performed algorithm better than the AHY algorithm. However, in some cases it still gave the same query response time as the AHY algorithm and therefore, overall the percentage difference varies.

The following graph illustrates the values from table 5.2:

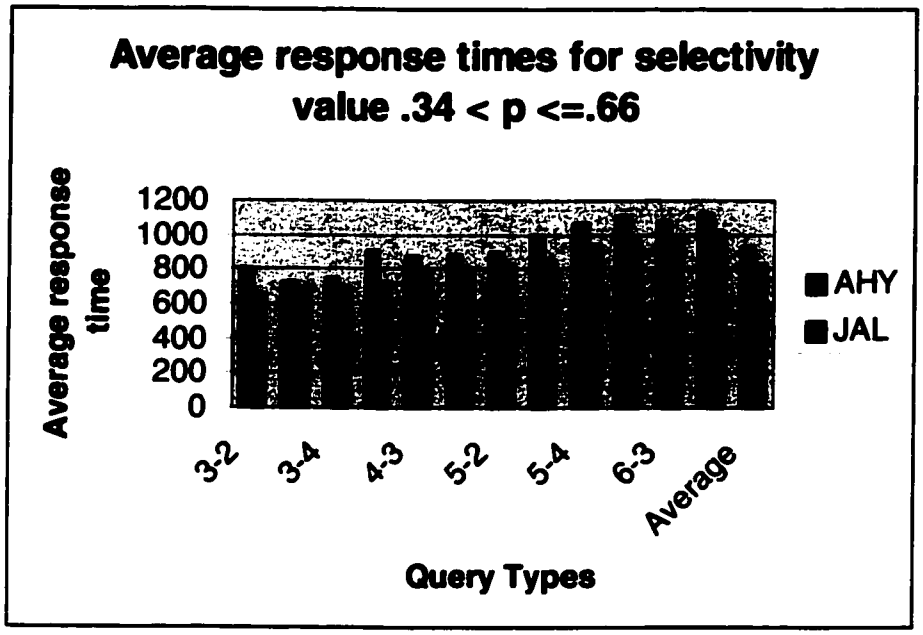



**Figure 5.2: Average response time with selectivity value .34 < p < = .66**

The query response time for the JAL dynamic algorithm on average is better when the selectivity values ranges between .34 < p < = .66. The average percentage improvement in the response time is 12.99 %.

| Type | AHY | JAL | Percentage Improvement |
|---|---|---|---|
| 3-2 | 1396.26 | 1289.02 | 8.25 |
| 3-3 | 1552.04 | 1311.29 | 18.36 |
| 3-4 | 1487.82 | 1324.28 | 12.35 |
| 4-2 | 1567.70 | 1354.65 | 15.27 |
| 4-3 | 1474.93 | 1395.27 | 5.71 |
| 4-4 | 1608.47 | 1437.68 | 11.88 |
| 5-2 | 1672.82 | 1470.36 | 13.77 |
| 5-3 | 1702.44 | 1531.25 | 11.18 |
| 5-4 | 1754.79 | 1554.29 | 12.90 |
| 6-2 | 1807.54 | 1571.23 | 15.04 |
| 6-3 | 1831.04 | 1612.12 | 13.58 |
| 6-4 | 1967.57 | 1721.41 | 14.30 |
| **Average** | 1668.83 | 1464.40 | 13.96 |

**Table 5.3: The percentage improvement in the response time with selectivity value .67 < p < = .99**

The above table shows the percentage time difference for the 12 different queries with the selectivity values .67 < p < = .99.

It was found that when the selectivity values are in the range of .67 - .99, in all the cases the JAL dynamic algorithm performed better than the AHY algorithm. The overall average percentage improvement in the response time is 13.96%.

The following graph illustrates the values from table 5.3:

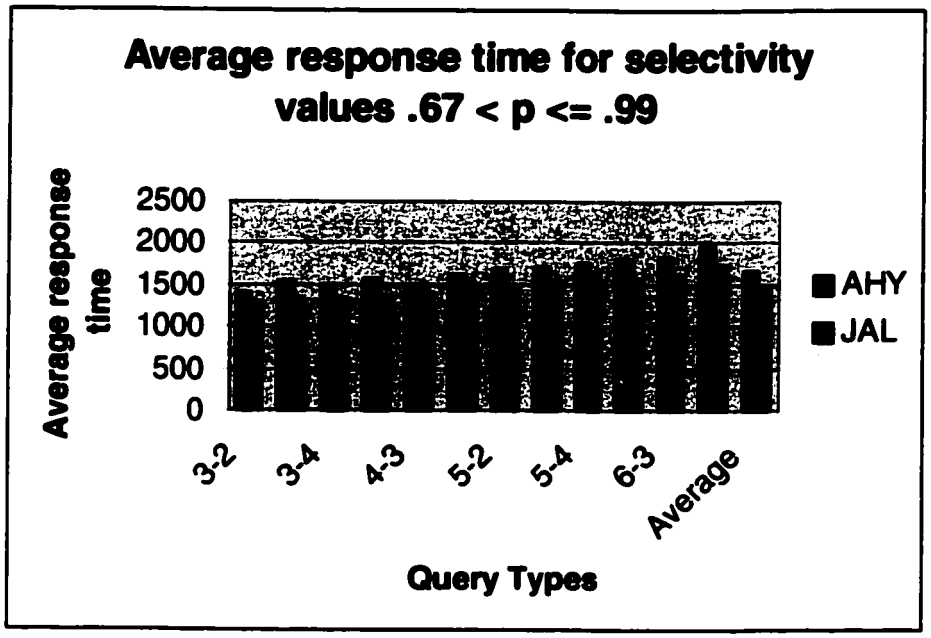



**Figure 5.3: Average response time with selectivity value .67 < p < = .99**

The query response time for the JAL dynamic algorithm on average is better when the selectivity values ranges between .67 < p < = .99. The average percentage improvement in the response time is 13.96 %.

## 5.3 CONTRIBUTION OF THE THESIS AND ITS EVALUATION

### 5.3.1 ADVANTAGES OF THE JAL DYNAMIC ALGORITHM

The following are some advantages of the JAL dynamic algorithm:

1. The JAL dynamic algorithm modifies the static AHY algorithm by making use of the profit concept. It calculates the cost and benefit of creating a candidate schedule for each joining attribute and generates a candidate schedule only if it is profitable.
2. The JAL dynamic algorithm gets rid of the redundancies, which increases the query response time.
3. It makes query processing dynamic and without adding extra monitoring or decision-making costs it makes the query retrieval adaptive.
4. It reduces the query response time for the higher selectivity values. This is very important because the higher the selectivity value, the lesser the chance is for a relation to get reduced, so it helps in overcoming the disadvantage of higher selectivity value.
5. It does not add any complexity to the existing heuristic.

### 5.3.2 LIMITATION OF THE JAL DYNAMIC ALGORITHM

1. When the selectivity values are $p <= .33$, the response time for both AHY and the JAL dynamic algorithms are the same. It does not make any difference except that it is adaptive in nature.
2. For the selectivity range between .34 - .66, sometimes the query response time for both the algorithms are the same. Therefore, the JAL dynamic algorithm is not able to identify which set of query types or factors lead to the same query response time. However, on average it performs better.
3. When the selectivity values are between the ranges .67 - .99, the JAL dynamic algorithm performs better than the AHY algorithm on average by 13.96 %. However, this is a slight improvement when we compare it with 2. Therefore,

after reaching a certain point, the performance improvement in the query response time is not a lot. However, the increase does exist.

# 6. CONCLUSIONS AND FUTURE WORK

## 6.1 CONCLUSION

A new algorithm, the JAL dynamic algorithm, is presented to make query optimization in distributed databases adaptive. It makes use of the concept of profit and ensures that the query response time are at least equal to or less than those achieved by the AHY heuristics.

We maintain that this approach of query optimization reduces query response time without adding extra time complexity and without having to provide extra data about the partial results or the relations. It also does not require any kind of monitoring as used in previous heuristics [BPR93]. It is a purely dynamic approach, which creates the query plans on the fly and returns the responds to the query response time with less than or equal the time achieved by previous heuristic.

The experiments using the statistics file, improves the performance. However, additional experiments can be conducted on very large relations to check if those percentage improvements are significant.

Extensive testing of the algorithms indicates that the JAL dynamic algorithm outperforms the AHY General algorithm.

## 6.2 FUTURE WORK

Some directions for the future work would be:

1. Identify factors that led to same query response time when the range of selectivity is between .34 and .66 and how can we make use of them to make the algorithm more adaptive.
2. Make use of business techniques and concepts such as "Marginal profit calculation" to enhance the performance of the JAL dynamic algorithm.
3. What will be the impact of the JAL dynamic algorithm when incorporated within object-relational databases?

4. How will the threshold value change when dealing with object-relational databases?
5. Incorporate other join techniques such as: Composite Join and Bloom Filter in the JAL dynamic algorithm to make more effective.
6. Remove redundancies and improvise the algorithm to incorporate the algorithm SERIAL part of the AHY algorithm.

# 7. REFERENCES

[ABK01] Atnafu, S.; Brunie, L.; Kosch, H., "Similarity-based operators and query optimization for multimedia database systems," IEEE Symposium on Database Engineering & Applications, pp. 346-355, 2001.

[AFT89] Laurent Amsaleg, Michael J. Franklin, Anthony Tomasic, "Dynamic query operator scheduling for wide area remote accesss," pp. 217-246, 1989.

[AHY83] P. Apers, A. Hevner and S. Yao, " Optimization algorithms for distributed queries," IEEE Transactions on Software Engineering, 9(1), pp. 51-60, 1983. (More information can also be obtained from Ph D. of A. Henver, "The optimization of query processing in distributed database systems, " Perdue University, 1980).

[AM91] J. Ahn and S. Moon," Optimizing joins between two fragmented relations on a broadcast local network," Info. Syst., vol. 16(2), pp. 185-198, 1991.

[BC81] P.A. Benstein and DM. W.Chiu, "Using semi joins to solve relational queries," J. Assoc. Comput., pp. 25-40, 1981.

[BE79] Babb, E., "Implementing a relational database by means of specialized hardware," ACM Transactions on Database Systems 4, March 1979.

[Bea95] W. T. Bealor, "Semijoin strategies for total cost minimization in distributed query processing," Master's Thesis, University of Windsor, 1995.

[Ber78] "Query processing in distributed database systems," Proceeding 3rd Berkeley Workshop on Distributed Data Management and Comput. Networks, Berkeley,

[BFM+00] Bouganim, L.; Fabret, F.; Mohan, C.; Valduriez, P., "Dynamic query scheduling in data integration systems," 16th International Conference on Data Engineering, pp. 425-434, 2000.

[BGW+81] P. Bernstein, N. Goodman, E.Wong, C. Reeve, and J. Rothnie, "Query processing in a system for distributed databases (SDD-1)," ACM Trans on Database Systems, vol. 6(4), pp. 105-128, 1981.

[BHC70] Bloom, B. H., Commum, "Space/time tradeoffs in hash coding with allowable errors," ACM 13(7), pp. 422, July 1970.

[BL82] P. Black and W. Luk, "A new heuristic for generating semi-join programs for distributed query processing," IEEE COMPSAC, vol. 581-588, 1982.

[BPGN81] Bernstrin, P., Goodman, N., "The power of natural semi joins," SIAM J. Comput., vol. 10(4), pp. 751-771, 1981.

[BPR90] P. Bodorik, J. Pyra, and J. Riordan, "Correcting execution of distributed queries," in Proc. Of 2nd Int. Symp. On Databases in Parallel and Distributed Systems, pp. 192-201, 1990.

[BRP92] P. Bodorik, J. Riordan, and J. Pyra, "Deciding to correct distributed query processing," IEEE Tans. On Knowledge and Data Engineering, vol. 4(3), pp. 253-265, 1992.

[CBH84] D. Chiu, P. Bernstein, and Y. Ho, " Optimizing chain queries in a distributed database system," Siam Journal of computing, vol. 13(1), pp. 116-134, 1984.

[CCY92] T.S. Chen, A.L.P. Chen, W.P. Yang, " Hash semi joins: A new technique for minimizing distributed query time," Proceedings of the 3rd Workshop on Future Trends of Distributed Computing Systems, pp. 325-330, 1992.

[CG94] Richard L. Cole, Goetz Graefe, " Optimization of Dynamic Query Evaluation Plans," ACM SIGMOD, May 1994.

[CGP86] Ceri, S., Gottlob, G., and Pelagatti, G., "Taxonomy and formal properties of distributed joins," Info., Systems, vol. 11(1), pp. 25-40, 1986.

[CH80] D. M. Chiu, Y. C. Ho, "A methodology for interpreting tree queries into optimal semi join expressions," Procedural ACM-SIGMOD, pp. 167-178, 1980.

[CH84] D. Chiu and Y. Ho, "Optimizing star queries in a distributed database system," in VLDB, pp. 959-967, 1984.

[CL80] A. Chen and V. Li, "A method for interpreting tree queries into optimal semi join expressions," in ACM SIGMOD, 1980.

[CL84] A. L. P. Chen, V. O. K. Li, "Deriving optimal star queries in a distributed query processing," IEEE INFOCOM, 1984.

[CL84] A.L.P. Chen and V.K Li, "Optimizing star queries in a distributed database system," Proceeding 10th Int. conference very large databases, 1984.

[CL84] L. Chen and V. Li, " Improvement algorithms for semi-join query processing programs in distributed database systems," IEEE Trans. On Computers, vol. 33(11), pp. 959-967, 1984.

[CL85] Arbee L. P. Chen and Victor O. K. Li, "An Optimal Algorithm for Processing Distributed Star Queries," IEEE Trans. On Software Engineering, vol.11 (10), 1985.

[CL89] J. S. J. Chen and V.O. K. Li, "Optimizing joins in fragmented database systems on a broadcast local network," IEEE Trans. Software Engineering, vol. 15(1), pp. 26-38, 1989.

[CL90] L. Chen and V. Li, " Domain-specific semi-join: a new operation for distributed query processing," Info. Sci., vol. 52, pp. 165-183, 1990.

[CLV84] Chen, L. Li, V., "Improvement algorithms for semi join query processing programs in distributed database systems," IEEE Trans. Comput., vol.c-33, 1984.

[CS01] Chiou, A.S.; Sieg, J.C., "Optimization for queries with holistic functions," Seventh International Conference on Database Systems for Advanced Applications, pp. 327 –334, 2001.

[CWY88] Cornell, D. W., and Yu, P. S., "Site assignment for relations and join operations in the distributed transaction processing environment," Proceedings of conference on Data Engineering, pp. 100-108, 1988.

[CY80] Cheung, T-Y, " A method for equi-join queries in distributed relational databases," IEEE Trans. On Computers, vol. 31(12), 1980.

[CY90a] M. S. Chen and P. S. Yu, "Using combination of join and semi join operations for distributed query processing," 10th International Conference on distributed computing systems, pp. 328-335, 1990.

[CY90b] M. S. Chen and P. S. Yu, "Using join operations as reducers in distributed query processing," Proc. Of the Second Int. Symp. On Databases in Parallel and Distributed Systems, 1990.

[CY93] M. Chen and P. S. Yu, "Combining join and semi-join operations for distributed query processing," IEEE Trans. On Knowledge and Engineering, vol. 5(3), pp. 534-542, 1993.

[CY94] M. S. Chen and P. S. Yu, "A graph theoretical approach to determine a join reducer sequence in distributed query processing," IEEE Trans. On Knowledge and Data Engineering, vol. 6, pp. 152-165, 1994.

[DH02] Deshpande Amol, Hellerstein Joseph M., ""Decoupled Query Optimization for Federated Database Systems," International Conference on Data Engineering,

March 2002. (The proceedings will be published in conference to be held on March 26, 2002 and therefore, the page numbers are unavailable currently). Website: http://db.cs.berkeley.edu/papers/ERL/erl01.html.

[DS01] Davis, K.B.; Sadri, F., " Optimization of schema SQL queries," International Symposium on Database Engineering & Applications, pp. 111-116, 2001.

[ER82] Epstein, R., " Query processing techniques for distributed, relational database systems," University Microfilms International, Ann Arbor, MI, 1982.

[ES80] R. Epstein and M. Stonebraker, "Analysis of distributed database processing strategies," Proc 6th International Conference On Very Large Databases, 1980.

[GGD+81] Gouda, M. G. and Dayal, U., "Optimal semi join schedules for query processing in local distributed database systems," ACM-SIGMOD, pp. 164-173, 1981.

[GL93] Glover, F. and M. Laguna, "Tabu Search," Chapter 3 in C, R. Reaves (ed.), Modern Heuristic Techniques for Combinatorial Problems, Blackwell Scientific publications, pp. 70-150, 1993.

[GM95] Bajan Groselj and Qutaibah M. Malluhi, "Combinatorial optimization of distributed queries," IEEE Trans. On Knowledge and Data Engineering, vol. 7(6), December 1995.

[Goo79] Goodman, "Query processing in a system for distributed organization," N. Technical Report, Computer Corporation of America, Cambridge, MA, 1979.

[Gra93] Goetz Grafefe, "Query Evaluation techniques for large databases," ACM Computing Surveys, vol. 25(2), June 1993.

[GS82] Goodman, N., and Shmueli, O., "The tree property is fundamental for query processing," ACM Symposium on Principles of Data Systems, 1982.

[GW89] Goetz Graefe, Karen Ward, "Dynamic query evaluation plans," ACM-SIGMOD, 1989.

[Hev80] A. Hevner, The optimization of query processing in distributed database systems. PhD thesis, Perdue University, 1980.

[Hou92] W. C. Hou et al, "Error- constrained count query evaluation in relation databases," SIGMOD, pp. 278-287, 1992.

[HWY85] Alan R. Hevner, O. Q. Wu and S. B. Yao , "Query optimization on local area networks," ACM Trans. Inf. Syst. 3(1), pp. 35 – 62, 1985.

[HY78] A. R. Hevner and S. B. Yao, " Optimization of data access in distributed systems," Computer Science Depts., Purdue University, Technical Report, TR281, 1978.

[HY79] A. R. Hevner and S. B. Yao, "Query processing in distributed database system," IEEE Trans. Software Engineering, vol. Se-5(3), 1979.

[Ioa96] Yannis E. Ioannidis, "Query Optimization," ACM Computing Surveys, vol. 28(1), March 1996.

[IW87] Ioannidis, Y.E. and E. Wong, "Query Optimization by Simulated Annealing," ACM-SIGMOD, pp. 9-22, 1987.

[JK84] Matthias Jarke, Jurgen Koch, "Query Optimization in Database Systems," Computing Surveys, vol. 16(2), June 1984.

[Kos00] Donald Kossmann, "The state of the art in distributed query processing," ACM Computing Surveys (CSUR), vol. 32(4), pp. 442-469, 2000.

[SJS00] Slivinskas, G.; Jensen, C.S.; Snodgras, R.T., "Query plans for conventional and temporal queries involving duplicates and ordering," International Conference on Data Engineering, pp. 547 –558, 2000.

[BFS00] Bandyopadhyay, S.; Fu, Q.; Sengupta, A., "A cyclic multi-relation semijoin operation for query optimization in distributed databases," Performance, Computing, and Communications Conference (IPCCC '00) pp. 101 –107, 2000.

[KOYU99] Kato, H.; Oyama, K.; Yoshikawa, M.; Uemura, S., "A query optimization for XML document views constructed by aggregations," International Symposium on Database Applications in Non-Traditional Environments (DANTE '99), pp. 189 – 196, 1999.

[KR87] H. Kang and N. Roussopoulos, "Using 2-way semi-joins in distributed query processing," in Proc. 3rd Int. Conference on Data Engineering, pp. 644-650, 1987.

[KS00] Donald Kossmann, Konrad Stocker, "Iterative Dynamic Programming: A new class of Query Optimization Algorithms," ACM Transactions on Database Systems (TODS), vol. 25(1), pp. 43-82, 2000.

[KTD01] Komaragiri, V.; Thornton, M.A.; Drechsler, R., "Application of a hardware synthesis technique for database query optimization," IEEE Pacific Rim Conference Communications, Computers and signal Processing, 2001 PACRIM, vol. 2, pp. 716-719, 2001.

[KYY+83] Kamabayshi, Y., and Yoshikawa, M., "Query processing utilizing dependencies and horizontal decomposition," ACM-SIGMOD, pp. 55-67, 1983.

[KYY+S82] Kambayshi, Y.,Yoshikawa, M., and Yagima, S., "Query processing in Distributed Databases using generalized semi joins," ACM-SIGMOD, pp. 151-160, 1982.

[LCC01] Jianzhong Li; Zhipeng Cai; Shuoying Chen, "Multi-weighted tree based query optimization method for parallel relational database systems," Cooperative Database Systems for Advanced Applications CODAS 2001 pp. 186 –193, 2001.

[LHC+85] Lu, H., and Carey, M., " Some experimental results on distributed join algorithms in a local area network," Proceedings of Conference on a very large database, pp. 292-304, 1985.

[Lia99] Yan Liang, "Reduction of collisions in bloom filters during distributed query optimization," Master's Thesis, University of Windsor, 1999.

[LN90] R. J. Lipton and J. F. Naughton, "Practical selectivety estimation through adaptive sampling," Proceedings of SIGMOD, pp. 1-11, 1990.

[LNA01] Lakshmanan, L.V.S., Shiri, N., "A parametric approach to deductive databases with uncertainity," IEEE Transactions on Knowledge and Data Engineering, vol. 13 (4), pp. 554-570, July-August 2001.

[LPP91] P. Legato, G. Paletta, and L. Palopoli, "Optimization of join strategies in distributed databases," Info. Syst., vol. 16 (4), pp. 363-374, 1991.

[LSC01] Chiang Lee; Chi-Sheng Shih; Yaw-Huei Chen , "Optimizing large join queries using a graph-based approach," IEEE Trans on Knowledge and Data Engineering, vol. 13 (2), pp. 298-315, March-April 2001.

[SJS01] Slivinskas, G.; Jensen, C.S.; Snodgrass, R.T., " A foundation for conventional and temporal query optimization addressing duplicates and ordering," Knowledge and Data Engineering, IEEE Transactions on, vol.13 (1), pp. 21 –49, 2001.

[LW86] Stéphane Lafortune and Eugene Wong, "A state transition model for distributed query processing," ACM Trans. Database Syst. , vol. 11(3), pp. 294-322, 1986.

[Ma97] X.Ma, "The use of bloom filters to minimize response time in distributed query optimization," Master's Thesis, University of Windsor, 1997.

[MB95] J. M. Morrissey and S. Bandyopadhyay, "Computer communication technology and its effects on distributed query optimization strategies," 1995.

[MB96] J. M. Morrissey and W. T. Bealor, "Minimizing data transfers in distributed query processing: A Comparative Study And Evaluation," The Computer Journal, vol. 39(8), 1996.

[MCS88] Mannino, M. V., Chu, P. and Sager, T., "Statistical profile estimation in database systems," ACM Computing Survey 20(3), pp. 192-221, Sept. 1988.

[ME92] Priti Mishra and Margaret H. Eich, "Join Processing in relational database," ACM Computing Surveys, vol. 24(1), March 1992.

[MIH87] Masuyama, S., Ibaraki, T., Nishio, S., and Hasegawa, T., "Shortest semi join schedule for a local area distributed database system," IEEE Transaction for Software Engineering, se-13 (5), pp. 602-606, 1987.

[ML86] Mackert, L. F., and Lohman, G. M., "R* Optimizer: Validation and Performance Evaluation for Distributed Queries," Proceedings of Conference on very large databases, pp. 149-159, 1986.

[MM98] J. M. Morrissey and X. Ma, "Investigating response time minimization in distributed optimization," Proceedings of ICCI, 1998.

[MO97] J.M. Morrissey and W. K. Osborn, "Experiments with the use of reduction filter in distributed query optimization," 9th IASTED International Conference on Parallel and Distributed Computing and Systems, pp. 327-330, 1997.

[Mor01] J. M. Morrissey, "Notes on Distributed Query Optimization," University of Windsor, 2001.

[Mor96] J. M. Morrissey, "Reduction filters for minimizing data transfers in distributed query optimization," Proceedings of the 1996 Canadian Conference on Electrical and Computer Engineering, pp. 198-201, 1996.

[MSW75] G. M., Stonebraker and E. Wong, "INGRES: A relational database system," Proceeding NCC, AJIPS Press, Montvale, N.J., 1975.

[MSW88] Mikkillineni, K. P., and Su, S. Y. W. " An evaluation of relational join algorithms in a pipelined query processing environment," IEEE Transaction on Software Engineering, vol. 14(6), pp. 838-848, 1988.

[Mul90] J. K. Mullin, "Optimal semi joins for distributed databases systems," IEEE Trans. On Software Engineering, pp. 558-560, vol. 16(5), 1990.

[Mul93] J. K. Mullin, "Estimating the size of a relational join," Information Systems, 18(3), pp. 189-196, 1993.

[NW01] Julio C. Navas and Michael Wynblatt, "The network is the database: data management for highly distributed systems," Proceedings of the 2001 ACM SIGMOD international conference on Management of Data on Management of data, pp. 544 – 551, 2001.

[Osb96] W. K. Osborn, "Distributed query optimization using bloom filters," Report 60-491, University of Windsor, 1996.

[Osb98] Wendy K. Osborn, "The use of reduction filters in distributed query optimization," Master's Thesis, University of Windsor, 1998.

[OV91] M.T.Ozsu and P. Valduriez, "Principles of Distributed Database Systems," Prentice-Hall, chapter 4, pp. 79, 1991.

[PC90] W. Perrizo and C. Chen, "Composite semi join in distributed query processing," Info. Sci., pp. 50, 1990.

[PLH89] Perrizo, W., Lin, J. Y. Y., and Hoffman, W., " Algorithms for Distributed Query Processing in Broadcast Local area Networks," IEEE Trans. Know and Data Eng., vol. 1(2), pp. 215-225, 1989.

[PS01] Plodzien, J.; Subieta, K., "Static analysis of queries as a tool for static optimization," Database Engineering & Applications International Symposium on, pp. 117-122, 2001.

[SMBBF01] Shapiro, L.; Maier, D.; Benninghoff, P.; Billings, K.; Fan, Y.; Hatwal, K.; Wang, Q.; Zhang, Y.; Wu, H.-M.; Vance, B., "Exploiting upper and lower bounds in top-down query optimization," Database Engineering & Applications, 2001 International Symposium on, pp. 20-33, 2001.

[GSV01] Gupta, A.; Sudarshan, S.; Vishwanathan, S., "Query scheduling in multi query optimization," International Symposium on Database Engineering & Applications, pp. 11-19, 2001.

[RK91] N. Roussopoulos and H. Kang, "A pipeline n-way join algorithm based on the 2-way semi-join program," IEEE Trans. On Knowledge and Data Engineering, vol. 3(4), pp. 486-495, 1991.

[RLM87] Richardson, J. P., Lu, H. and Mikkilineni, K., "Design and evaluation of parallel pipelined join algorithms," ACM-SIGMOD, pp. 399-409, 1987.

[RM97] S. Rho, S. T. March, "Optimizing distributed join queries: A genetic algorithm approach," Annals of Operation Research, vol. 71, pp. 199-228, 1997.

[RRL97] Celso C. Riberio, Claudio D. Ribeito and Rosana S. G. Lanzelotte, "Query Optimization in distriubuted relational databases," Journal of Heuristics, vol. 3, pp. 5-23, 1997.

[Seg86] Arie Segev, "Optimization of join operations in horizontally partitioned database systems,"ACM Trans. Database Syst., vol. 11, pp. 48-80, 1986.

[Sel88] Timos K. Sellis, "Multiple-query optimization," ACM Trans. Database Syst. 13(1), pp. 23 – 52, 1988.

[SG88] A. Swami and A. Gupta, "Optimization of large join queries," ACM-SIGMOD conference, pp. 8-17, 1988.

[SW91] Dennis Shasha and Tsong-Li Wang, "Optimizing equijoin queries in distributed databases where relations are hash partitioned," ACM Trans. Database Syst. 16(2), pp. 279-308, 1991.

[Swa89] Swami, A., "Optimization of large join queries: Combining heuristics and combinatorial techniques," ACM-SIGMOD Conference, pp. 367-376, 1989.

[TC92] Judy C.R. Tseng, Arbee L. P. Chen, "Improving Distributed Query Processing by Hash Semi joins," Journal of Information Science and Engineering 8, pp. 525-540, 1992.

[VYV+84] Varol, Yl and Vrbsky, Sv., "Distributed Query Processing strategy for redundant data," Proc. 4th International Conference on Distributed Computing Systems, May 1984.

[WC93] C. Wang and M. Chen, "On the complexity of distributed query optimization," technical report, IBM Technical Report RC 18670, 1993.

[WC96] C. Wang and M. S Chen, "On the complexity of distributed query optimization," IEEE Trans. On Knowledge and Data Engineering, vol. 8 (4), August 1996.

[WLC91] C. Wang, V.O. K. Li, and A. L. P. Chen, "Distributed query optimization by one-shot fixed-precision semi-join execution," Proceedings 7th International Conference on Data Engineering, pp. 765-763, 1991.

[Won77] E.Wong, "Retrieving dispersed data from SDD-1: A system for distributed databases," Proc. 2nd Bekeley Workshop Distributed Data Management and Computing Networks, pp. 217-235, May 1977.

[YB79] Yao, S. B., "Optimization of query evaluation algorithms," ACM Transaction Database System, vol. 4(2), pp. 133-155, 1979.

[YBC77] Yao, S.B, Commun., "Approximating block accesses in database organization," ACM, vol. 20(4), pp. 260, 1977.

[YC84] C. T. Yu and C. C. Chang, "Distributed query processing," ACM Comput. Surv. 16(4), pp. 399 – 433, Dec. 1984.

[YCC83] Yu, Ct and Chan, CC., "On design of a query processing strategy in a distributed database environment", ACM-SIGMOD, p. 30-39, 1983.

[YCT+85] Yu, C. T, Chang, C. C., Templeton, M. Brill, D., and Lund, E., " Query processing in a fragmented relational database system: Mermaid," IEEE Trans. Software Engineering, se-11 (8), pp. 795-810, 1985.

[YGZ+87] Yu, C. T., Guh, K-C., Zhang, W., Templeton, M., Brill, D., and Chen, A.L. P., "Algorithms to process distributed queries in fast local networks," IEEE Trans. Comput. C-36(10) pp. 1153-1164, 1987.

[YHL89] Yoo, Ho and Lforutne, S., " An intelligent search method for query optimization by semi joins," IEEE Trans. On Know. And Data Eng., pp. 226-237, vol.1 (2), June 1989.

[YLO80] C. T. Yu, K.Lam, M. Z. Ozsoyoglu, "Distributed query optimization for tree queries," Dep. Information Engineering, University of Illinois at Chicago Circle, 1980.

[YOK84] C.Yu, Z. Ozsoyoglu, and K. Kam, "Optimization of distributed tree queries," J. Comp. Sys. Sci., vol. 29(3), pp. 409-445, 1984.

## VITA AUCTORIS

| | |
|---|---|
| Name | Lubna Sachwani |
| Year of Birth | 1978 |
| Place of Birth | Karachi, Pakistan |
| Education | 1. GCE Advanced Level, 1996<br>2. B. Comm. Honours Business Administration, University of Windsor, 2000<br>3. Master of Science in Computer Science, University of Windsor, 2002. |