2004

An evaluation of a pipeline planner for a filter based algorithm.

Xiaobai Cao
University of Windsor

An Evaluation of a Pipeline Planner for a Filter Based Algorithm

by

Xiaobai Cao

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

2004

Your file Votre référence
ISBN: 0-612-92464-5
Our file Notre référence
ISBN: 0-612-92464-5

Xiaobai Cao 2004

©All Rights Reserved

ABSTRACT

Distributed query processing is one of the technical problems that need to be solved in Distributed Database Management Systems. Query Processing deals with designing algorithms that analyze queries and converts them into a series of data manipulation operations. The problem is how to decide on a strategy for executing each query over the network in the most cost effective way.

Through the past years, the research focus in distributed query processing has been on how to realize join operations with different operators such as Semi-join, Two-way Semijoin, and Pipeline N-Way joins. However, these operations will be executed sequentially, which may increase the data transfer cost.

A new algorithm, filter based pipeline N-way join algorithm, is presented to reduced data transfer cost. It makes use of filter concept and ensures the lower data access cost. This algorithm has three phases. Phase One: Use bloom filter to do forward semijoin and build tuple connectors. Phase Two: Do backward semijoin and build pipeline cache planner. Phase Three: Send Pipeline Cache Planner to query site.

The main goal for this new algorithm is to reduce data transfer cost while maintain low I/O cost as pipeline N-way join algorithm.

To my Family and Friends

ACKNOWLEDGEMENTS

I am very happy to take this opportunity to thank people who helped me a lot during the whole process of my graduate study and towards the achievement of my thesis.

I would like to express my sincere gratitude and appreciation to my supervisor Dr. Joan Morrissey. The work cannot be accomplished without her extensive guidance. She is very nice, kind and supportive. Whenever I have some problems in either study or daily life, she will help me without any hesitation.

I will always be indebted to all my teachers and my thesis committee members (in alphabetical order): Dr. Myron Hlynka, Dr. Jianguo Lu, Dr. Joan Morrissey, and Dr. Alioune Ngom for their guidance and a lot of support. Their valuable advices gave my thesis professional look.

I can never forget the love, support, help, guidance given by my parents and brother. Without them, I would never have today…

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

Distributed database system technology is one of the major recent developments in the database systems area. In this chapter, we give a brief introduction on some important concepts and notations.

1.1 What is a Distributed Database System?

A Distributed Database System (DDBS) can be defined as a collection of multiple, logically interrelated connected databases over a computer network. It is not a system where the database resides at only one site, but is distributed among a number of sites. To form a DDBS, files should not only be logically related, but there should be structure among the files, and access should be via a common interface.

1.1.1 Advantages and Problems of DDBSs

Based on the features of distributed Database Management System (DBMS), it has a better performance than separated databases. In detail, a distributed DBMS fragments the conceptual database, enabling data to be stored in proximity to its points of use (also called data localizing). This has three advantages:

1. Reduce Overhead

   Since each site handles only a portion of the database, contention for CPU and I/O services is less than in centralized databases. Suppose, you want to do query, and if you do it within local centralized database, the I/O will be much less than do it in distributed database.

2. Localization

   Localization can reduce remote access delays that are usually involved in wide area networks. Most distributed DBMS are structured to obtain maximum benefit from data localization. Full benefits of reduced contention and reduced communication overhead can be obtained only by a proper fragmentation and distribution of the database.

3. Provide intro-query and inter-query

   Intra-query parallelism is achieved by executing multiple queries at the same time. Inter-query parallelism means to break up a single query into a number of subqueries each of which is executed at a different site, accessing a different part of the distributed database.

Though DDBS provides some benefits, it produces problems too. Here are some problem areas that exist in DDBSs.

1. Distributed Query Processing

   Query processing deals with designing algorithms that analyze queries and convert them into a number of data manipulation operations. The problem is how to decide on a strategy which can minimize the cost.

2. Distributed Concurrency Control [OV99b]

   Concurrency control involves the synchronization of accesses to the distributed database, so that the integrity of the database is maintained. It deals with the isolation and consistency properties of transactions. With no doubt, concurrency control is one of the most extensively studied programs in DDBS field.

3. Distributed Database Design

   Distributed database design deals with how the databases and applications are placed across the sites. Basically, it has two alternative ways: partitioned and replicated. In a partitioned scheme, the database is divided into a number of disjoint partitions each of which is placed in a different site. Replicated design can be either place the entire database in each site or each partition of the database is stored at more than one site.

1.2 Query Optimization Processing

Query optimization refers to the process of producing a query execution plan (QEP) which represents an execution strategy for the query. The selected plan minimizes an objective cost function. A query optimizer has three components: a search space, a cost model and a search strategy (see Figure 1.1).

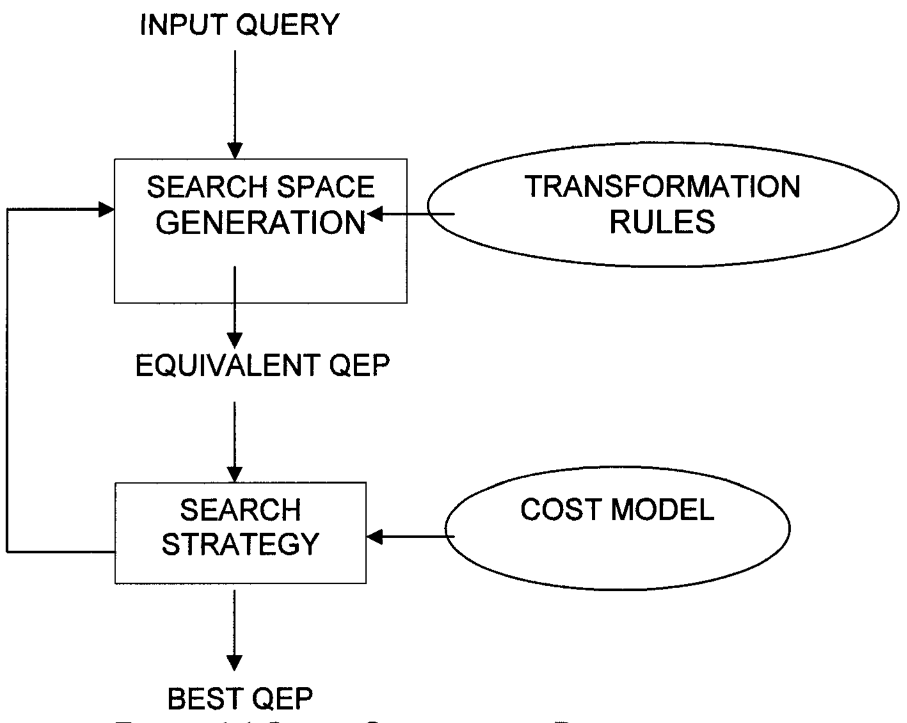

FIGURE 1.1 QUERY OPTIMIZATION PROCESS

The search space is the set of alternative execution plans to represent the input query, where query execution plans define the order in which the operations are executed. These plans are equivalent, in the sense that they yield the same result but they differ in the execution order of operations and the way these operations are implemented, and therefore on performance.

The cost model includes cost functions to predict the cost operators, statistics and base data and formulas to evaluate the sizes of intermediate results. To be accurate, the cost model must have good knowledge about the distributed execution environment.

The search strategy uses the cost model to explore the search space and selects the best plan. It indicates which plans are examined and in which order.

The selection of the optimal query processing strategy normally requires the prediction of execution cost of the alternative candidate orderings prior to actually executing the query. The execution cost is defined as a weighted combination of I/O, CPU, and communication costs. To simplify the problem, the local processing cost (I/O and CPU costs) is ignored, and it is assumed that the dominant cost is communication. The problem is NP-hard in nature [OV99a], and the approaches are usually heuristic.

Query Processing Steps

Given a distributed query, how to generate an optimal processing strategy has been of great interest to a number of researchers [AHY78a, AHY78b, CC83, J82, LCC82, PW82, PWC+81, WE77]. In these papers, a given distributed query is processed through the following three phases:

1. Local processing phase

   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.

2. Reduction Phase

   Some operators can be used to reduce cost, i.e. Semijoin. A semijoin program (a sequence of semijoins [PW81]) is generated by a query optimization algorithm. The semijoin program reduces the sizes of relations and/or intermediate results in a cost-effective way and thus, minimizes the total amount of data transmission.

3. Final query processing phase

   The final query-processing site is selected and the reduced relations and/or intermediate results are sent to that site. The final processing phase performs an N-way join either at the query site or in another site on behalf of the query site.

In this framework, the core phase is the reduction phase, and its primary concern is to generate the most efficient semijoin program to reduce the transmission cost. Several algorithms have been proposed, such as for simple queries [AHY78a], tree queries [CKM80, DY80], chain queries [DPY84], and star queries [CL84a, CL84b].

1.3 Organization of the thesis

This thesis consists of five chapters. Chapter 1 gives a brief introduction to distributed database systems and its query optimization process. Some relevant background information such as Joins, Semijoin, Two-way semijoin, Pipeline N-way join, and Filter based algorithms will be discussed in Chapter 2. In Chapter 3, new algorithm H will be proposed. An illustrated example shows how the algorithm H works. Chapter 4 gives the experimental system and evaluation results. Finally, in Chapter 5, conclusion and future work are discussed.

Chapter 2

Background Review

In distributed query processing, many approaches use join, semijoin and some other algorithms. This chapter provides background information on these algorithms. An in depth look is taken at some strategies.

2.1 Assumptions and Notation

In this chapter, a number of algorithms will be discussed and all of them are based on the following assumptions:

- A distributed relational database management system with a number of independent nodes distributed geographically and connected via a point-to-point network.

- Each node has local processing and storage capabilities.
- The relations are distributed amongst the nodes and all nodes can access all data.
- Only selected-projection-join (SPJ) queries are considered.
- A uniform and independent distribution of attribute values is assumed.

During query processing, certain information is available for use in constructing a strategy:

For each relation R_i, we have the following:

- R_{ij} stands for attribute j in Relation i
- $| R_j |$ is the number of tuples in the relation.

For each projection d_{ij}, the projection of relation i over attribute j, we have the following:

- $| d_{ij} |$ is the cardinality of the projection of relation i over attribute j.
- $| D (d_{ij}) |$ is the cardinality of the domain for attribute d_{ij}. It is the number of possible values in the domain, not the number of the actual values occurring in the database. Besides, we assume that the cardinality is finite and known.
- $\rho(d_{ij})$ is the selectivity of the projection d_{ij}, which is the portion of tuples participating in the join. It is commonly defined as $| d_{ij} | / | D (d_{ij}) |$.
- Domain is the set of allowable values for the attribute.
- Selectivity is the ratio of distinct attribute values over the attribute domain size.

Here is one example. Suppose we have relation R_1 with domain 100. It is shown as Figure 2.1.

A	B	C
1	3	6
2	2	4
3	3	8
4	4	65
5	1	8
6	4	4

FIGURE 2.1 RELATION R_1

Based on definition, R_{11} refers to attribute A, R_{12} refers to attribute B, and R_{13} refers to attribute C.

The projection of attribute A are 1, 2,3,4,5 and 6, therefore $| d_{11} | = 6$; the projection of B are 3,2,4,1 so $| d_{12} | = 4$; same as attribute C, its projection is 6, 4,8,65 so $| d_{13} | = 4$

Since we assume the domain size for relation R is 100, therefore we can say

$| D (d_{11}) | = | D (d_{12}) | = | D (d_{13}) | = 100$

By definition, we know the selectivity can be got by $| d_{ij} | / | D (d_{ij}) |$. So,

$\rho(d_{11}) = | d_{11} | / | D (d_{11}) | = 6/100 = 0.06$

$\rho(d_{12}) = | d_{12} | / | D (d_{12}) | = 4/100 = 0.04$

$\rho(d_{13}) = | d_{13} | / | D (d_{13}) | = 4/100 = 0.04$

2.2 Join Algorithm

JOIN is Cartesian product followed by selection. It joins two tables together on the basis of common values in a common column.

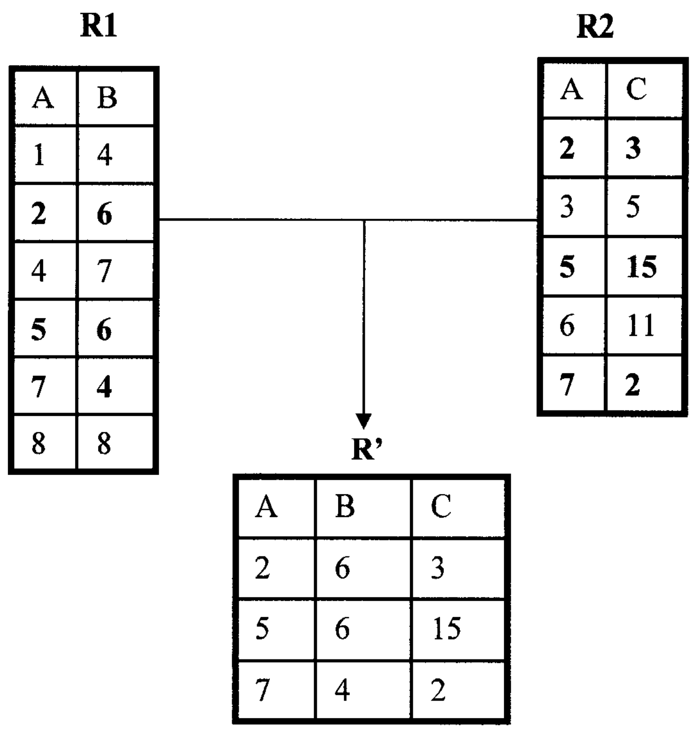

FIGURE 2.2 JOIN ALGORITHM

Suppose we have two relations R1 and R2. R1 has two attributes A and B. R2 also has two attributes A and C. We can see both relations have the same attribute A. Therefore the result of joining these two relations will have three attributes A, B and C. The transmission cost is the whole size of relation R1 and R2. So, when we migrate from centralized database systems to distributed database systems, the Join operator becomes the most costly process.

Ordering of joins is an important aspect of centralized query optimization. Two basic approaches exist to order joins in fragmented queries. One tries to optimize the ordering of joins directly, whereas the other replaces joins by combinations of semijoins in order to minimize communication costs. Distributed INGRES and R* algorithms are representative of algorithms that use joins rather than semijoins [OV99c].

2.2.1 INGRES Algorithm

INGRES uses a dynamic optimization algorithm [WY76] that recursively breaks up a calculus query into smaller pieces. It combines the two phases of calculus-algebra decomposition and optimization. A query is first decomposed into a sequence of queries having a unique relation in common. Then each monorelation query is processed by a "one-variable query processor (OVQP)". The OVQP optimizes the access to a single relation by selecting, based on the predicate, the best access method to that relation. For example, if the predicate is of the form <A = value>, an index available on attribute A would be used. However, if the predicate is of the form < A ≠ value >, an index on A would not help, and sequential scan should be used.

The query optimization algorithm of Distributed INGRES is derived from the algorithm used in centralized INGRES [OV99b]. Therefore, it consists of dynamically optimizing the processing strategy of a given query. The objective function of the algorithm is to minimize a combination of both the communication time and response time. However, these two objectives may be conflict. For example, increasing communication time may well decrease response time. Thus, the function can give a great weight to one or the other. Note this algorithm ignores the cost of transmitting the data to the result site. And the algorithm takes the advantage of fragmentation but only horizontal fragmentation is handled for simplicity.

2.2.2 R* Algorithm [SK+76]

System R performs static query optimization based on the exhaustive search of the solution space. The input to the optimizer of System R is a relation algebra tree resulting from the decomposition of an SQL query. The output is an execution plan that implements the "optimal" relational algebra tree.

The distributed query optimization algorithm of R* is a substantial extension of the techniques developed for System R's optimizer. It uses a complicated approach where an exhaustive search of all alternative strategies is performed in order to choose the one with the least cost. Although predicting and enumerating these strategies is costly, the overhead of exhaustive search is rapidly amortized if the query is executed frequently. Query compilation is a distributed task in R*, coordinated by a master site, where the query is initiated. The optimizer of the master site makes all intersite decisions, for example the selection of the execution sites, parts of the relation, and the method for transferring data. The apprentice sites, which are the other sites that have relations involved in the query, make the remaining local decisions and generate local access plans for the query. The objective function of the System R*'s optimizer is the general total time function, including local processing and communication costs.

2.2.3 SDD-1 Algorithm [BGW+81]

The query optimization of SDD-1 is derived from an earlier method called hill-climbing algorithm [WE77], which has the distinction of being the first distributed query-processing algorithm. In this algorithm, refinements of an initial feasible solution are recursively computed until no more cost improvement can be made. This algorithm is quite general in that it can minimize the arbitrary objective function, including the total time and response time.

2.3 Semijoin Algorithm

Semijoin has become the most popular operator recently to replace the Join operator, since it reduces the relation sizes and minimizes the cost of transmission. Many algorithms based on semijoin have been developed to process distributed semijoins [AHY78a, AHY83, BGW+81, C82, CBY84, CGS79, CL84b, EM78, GS82, KTY82, P84, P85, PM79, SY80, VG84, YC84]. The semijoin acts as a size reducer for a relation much as a selection does.

The steps to do the semijoin are [M81, W82]:

1. Send the projection $R_i[A]$ from site i to j.
2. Reduce R_j by eliminating tuples whose attributes A are not matching any value in $R_i[A]$.

Step one is the cost of the semijoin, and step two returns the benefit of the semijoin.

Cost and Benefit of Semijoin

Consider a semijoin $R_i \ltimes_A R_j$ when R_i and R_j are at different sites. Let the transmission cost be one per data unit transmitted. Then, the *cost* of the semijoin is $S(R_i[A])$, where

S(R) is the size of relation R. Suppose this semijoin reduces R_j to R_j'. Then the *benefit* is $S(R_j)$-$S(R_j')$[S88]. A cost effective semijoin is a semijoin whose benefit exceeds the cost.

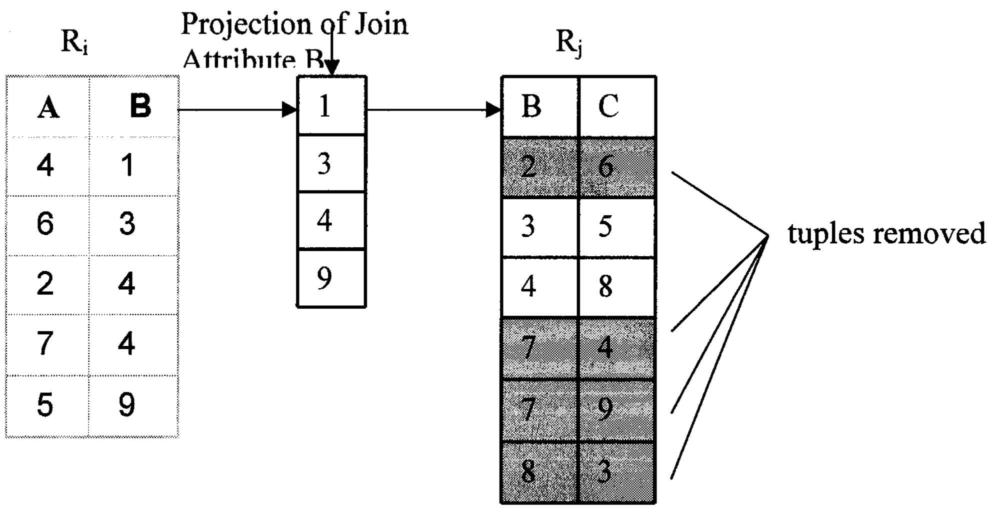

FIGURE 2.3 SEMI-JOIN

Here is one example, see Figure 2.3. There are two relations R_i and R_j. R_i is in site i, and R_j is in site j. The join attribute of relation R_i and R_j is B, therefore, we first project the attribute B over site i, and then transfer them to site j to reduce the relation R_j.

Here, we can see the cost of using the Initial Feasible Solution is 10 units, 5 units in A plus 5 units in B. The cost of the semijoin is 4 units, since only 4 units have been transferred. The benefit is 12-4= 8, which is the original relation with 12 units minus the reduced relation 4 units. Since cost is less than the benefit we can say that it's a cost-effective semijoin. Therefore, we can say that this semijoin is cost effective.

2.4 Two-way Semi-join Algorithm

In distributed query processing, a semijoin is used to reduce the relation which can minimize the total amount of data being transferred. The two-way semijoin is an extension of semijoin, for more cost effective distributed query processing. The two-way semijoin is compared to the semijoin in terms of the reduction power and the propagation of reduction effects.

Formally, the result of a two-way semijoin of R_i and R_j on attribute A is a set of two relations $\{R_i', R_j'\}$ where R_i' (R_j') is the projection on the attributes of R_i (R_j) of the join $R_i \bowtie_A R_j$. Let us denote the two-way semijoin of R_i and R_j on attribute A by $R_i \bowtie\!\!\!\!\bowtie\ {}_A R_j$. Then,

$R_i \bowtie\!\!\!\!\bowtie\ {}_A R_j = \{R_i \ltimes_A R_j, R_j \ltimes_A R_i\}$

<u>The steps to do the 2-way semijoin are</u> [KR91]:

1. Send the projection $R_i[A]$ from site i to j.
2. Reduce R_j by eliminating tuples whose attributes A are not matching any value in $R_i[A]$. This is the forward reduction of the semijoins. During the forward reduction of R_j, partition $R_i[A]$ into $R_i[A]_m$ and $R_i[A]_{nm}$ where $R_i[A]_m$ is the set of values in $R_i[A]$ which match one of $R_j[A]$, and $R_i[A]_{nm}$ is $R_i[A] - R_i[A]_m$.
3. Send either $R_i[A]_{nm}$ or $R_i[A]_m$ whichever is less in size from site j back to i.
4. Reduce R_i using either $R_i[A]_{nm}$ or $R_i[A]_m$. If $R_i[A]_m$ is used, then tuples whose attribute A are not matching any of $R_i[A]_m$ will be eliminated. If $R_i[A]_{nm}$ is used, then tuples whose attribute A are matching one of $R_i[A]_{nm}$ are eliminated.

We can see that the first two steps are exactly the same as semijoin, but how can this improve the query processing?

Cost and benefit of two-way semijoin

Step 2 and step 4 will result in the benefit for the two-way semijoin. Therefore, the benefit would be: $[S(R_i) - S(R_i')] + [S(R_j) - S(R_j')]$

The cost of the query processing is during the step of 1 and 3. The total cost woule be: $S(R_i[A]) + S(R_j'[A]) = S(R_i[A]) + S(R_i[A]_m)$

Same as semijoin, a cost-effective two-way semijoin is a two-way semijoin whose benefit exceeds the cost.

Now we use the same sample queries as semijoin, but apply two-way semijoin algorithm this time. Here is the example of two-way semijoin:

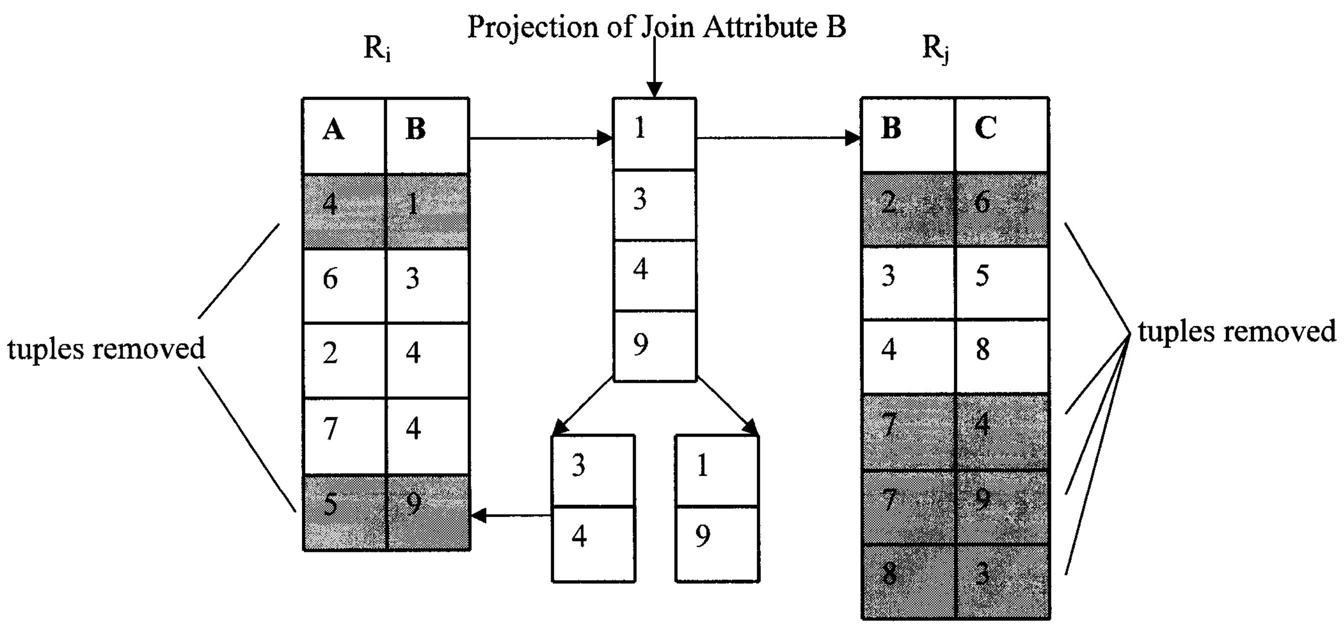

FIGURE 2.4 TWO-WAY SEMI-JOIN

We have two relations R_i and R_j at two different sites. R_i has two attributes A and B. R_j also has two attributes B and C. Therefore, the join attributes for R_i and R_j is B. The first step is to project the join attribute B, therefore 4 units (cost) have been transferred from site i to site j. Then, by comparing the value of attribute B in site j, four tuples, which are 8 units (benefit), will be removed. We can see that, up to this step, they are exactly the same as semijoin. Step 3 is to do the backward semijoin. The projection of attribute B in site j will be divided into two parts, one will match the projection of attribute B in site j, and the rest will be put in the other. After that, one of them, which are less, will be sent back to site i to reduce the relation R_i. Therefore, in the above example, we can see that two units 3, 4 (cost) will be sent back to site i, and 4 units (benefit) in relation R_i are reduced.

The cost of the above example is: $4 + 2 = 6$ units. The benefit is $8+4 = 12$ units. Since the benefit is greater than the cost, we can say that it's a cost-effective two-way semijoin.

Some existing heuristic algorithms based on semijoin [ACV+84, AHY83] can be modified by replacing the semijoin with two-way semijoin. The query processing strategy generated by a heuristic algorithm is, in general, suboptimal. The properties that an optimal semijoin program should possess have been studied [AR91] and the algorithms for improving semijoin programs have been proposed [A79, B70, BR88b, J82, RK91, Y87]. Most of these improving techniques for semijoin programs are applicable to query processing strategies using two-way semijoin as well.

Cost and Benefit of Two-way Semijoin

Step 2 and step 4 will result in the benefit for the two-way semijoin. Therefore, the benefit would be: $[S(R_i) - S(R_i')] + [S(R_j) - S(R_j')]$

The cost of the query processing is during the step of 1 and 3. The total cost would be: $S(R_i[A]) + S(R_j'[A]) = S(R_i[A]) + S(R_i[A]_m)$

Same as semijoin, a cost-effective two-way semijoin is a two-way semijoin whose benefit exceeds the cost.

2.5 Pipeline N-way Join Algorithm

In distributed database where data transmission is not the dominant cost factor in query processing, pipelining can be very efficient. The main goal of a pipeline algorithm is to eliminate the need for shipping, storing, and retrieving foreign relations and/or intermediate results on the local disks of the query site during the processing of an N-way join [KR91].

In order to understand the algorithm, we need to know the following notations.

- Tuple connectors: Some temporary relations constructed on the fly. It has tuple ID and real data.
- Tuple identifier (TID): For each relation R_i participating in the N way join, its tuple connector C_i is a projection of R_i on all the joining attributes and a tuple

identifier. The TID in C_i makes the correspondence between the tuples of R_i and C_i one-one. For example, @R12 stands for Relation 1 tuple 2.

- Pipeline cache planner: contains N-tuples of TID's of joinable tuples.

<u>Steps for Pipeline join algorithm</u>

1. Forward Reduction & Local Processing Phase
    - Site storing R_i receives from the site storing R_{i-1} the projection of the joining attribute needed for the forward reduction.
    - Tuple connector C_i is constructed
    - N tuple connectors reflect both forward reduction and local site processing.
2. Backward Reduction and Collection Phase
    - A site containing R_i receives from the site of R_{i+1} and C_{i+1} tuple connector and joins it with its own C_i.
    - The pipeline cache planner is an N-Way relation that holds the TID's of all joinable tuples of the N relations.
3. Pipeline Execution Phase
    - Pipeline cache planner is sent to the query site and used for synchronizing the tuple requests from the N sites in order to assemble the result.

Here is the example. We have 3 different relations in 3 different locations. @R is tuple ID.

R1

@R1	A	B	C	E
@R11	5	4	3	6
@R12	1	4	2	4
@R13	3	3	2	3

R2

@R2	A	D
@R21	1	4
@R22	5	6
@R23	7	5

R3

@R3	B	C	F
@R31	3	2	9
@R32	5	2	6
@R33	6	7	5
@R34	4	3	4

FIGURE 2.5 PIPELINE N-WAY JOIN ALGORITHM EXAMPLE

We can see that A, B, and C are join attributes. Phase one is to build C_i in local site and send C_i to R_{i+1}.

Step 1.1: Build C_1 Connector in R_1. Since R_1 is the first relation and there is no other relation can be forwarded to R_1, C_1 Connector is the projection of Join attributes in R_1. Since E is not a join attribute, it won't be included.

C1 Connector

@R1	A	B	C
@R11	5	4	3
@R12	1	4	2
@R13	3	3	2

FIGURE 2.6 TUPLE CONNECTORS C_1

Step 1.2: Send Projection of Join attributes in C_1 Connector to R_2. In this case, 9 units, which is the cost, will be forwarded to R_2.

Step 2.1: Build C_2 Connector in R_2, which is to join R_2 and C_1 Connector. Since D is not a join attribute, only A, B, C will be included.

C2 Connector

@R2	A	B	C
@R21	1	4	2
@R22	5	4	3

FIGURE 2.7 TUPLE CONNECTORS C_2

Step 2.2: Send Projection of Join attributes in C_2 Connector to R_3. In this case, 6 units will be forwarded to R_3.

Step 3.1: Build C_3 Connector in R_3, which is to join R_3 and C_2 Connector. Since F is not a join attribute, it won't be included in C3 Connector.

C3 Connector

@R3	A	B	C
@R34	5	4	3

FIGURE 2.8 TUPLE CONNECTORS C_3

Step 3.2: Since all the relations have been processed. This is the end of Phase one. In phase one, total cost is 9+6=15units.

During the next phase, it is a backward reduction, and pipeline cache planner (PCP) will be made.

Step 4.1: To build C3' Connector in site R3. Since there is no C4' Connector, C3' Connector will the same as C3 Connector.

C3' Connector

@R3	A	B	C
@R34	5	4	3

FIGURE 2.9 TUPLE CONNECTORS C_3'

Step 4.2: To form PCP3. Since there is only one relation, PCP contains one attribute which is the tuple ID.

@R3

@R3
@R34

FIGURE 2.10 PCP3

Step 5.1: To build C2' Connector in site R2. That is C3' Connector join C2 Connector. C3' Connector has one tuple and C2 Connector had two tuples. Comparing the value of join attributes A, B, and C' projection, only the second tuple in R2 will be remained which is the result of C2' Connector. The cost is 4 (3 units plus 1 tuple ID).

C2' Connector

@R2	A	B	C
@R22	5	4	3

FIGURE 2.11 TUPLE CONNECTORS C_2'

Step 5.2: To form PCP2.

@R2

@R2	@R3
@R22	@R34

FIGURE 2.12 PCP2

Step 6.1: To build C1' Connector in site R1. That is C2' Connector join C1 Connector. C2' Connector has only one tuple, and C1 Connector has three tuples. However, by matching the join attributes, only the first tuple in C1 Connector will be in C1' Connector. The cost is 4 (3 units plus 1 tuple ID).

C1' Connector

@R1	A	B	C
@R11	5	4	3

FIGURE 2.13 TUPLE CONNECTORS C_1'

Step 6.2: To form PCP1.

@R3

@R1	@R2	@R3
@R11	@R22	@R34

FIGURE 2.14 PCP1

Finally, The pipeline cache planner will be sent to the query site. In this example, the total cost is 2 + 4 + 6 = 12 units. Therefore, the total cost for all three phases is 15 + 8 + 12 = 35 units.

Now, let's check if the tuple ID in PCP is correct or not. In PCP1, it has one tuple with value of @R11, @R22, and @R34.

R1

@R1	A	B	C	E
@R11	5	4	3	6

R2

@R2	A	D
@R22	5	6

R3

@R3	B	C	F
@R34	4	3	4

FIGURE 2.15 REDUCED RELATIONS

If ship all the relations R1, R2, and R3 to query site and join them there. We will get the same result as shown in figure 2.15. Therefore, we can say we can get the same results as we choose IFS.

The main advantages of this algorithm are [Y85]:

- No intermediate results are generated.
- Tuple connectors, which are smaller in size. It means that we do not have to store the reduced relations. We save space and transmission costs are reduced.
- The original relations are only accessed once during the algorithm.
- No reduced relations are transferred to the query site.

Caching some or all of the delivered tuples can optimize the pipeline planner. Optimal caching for the planner is equivalent to optimal materialization of views stored in ViewCaches [BS92]. Elsewhere, it has been shown that optimal materialization is NP-complete [CC83]. Optimization of the planner is dependent on the availability of memory buffers and the sophistication of the algorithms.

2.6 AHY Algorithm

For a special class of simple queries, Hevner and Yao developed algorithm PARALLEL and SERIAL [AHY83] that find strategies with, respectively, minimal response time and total time. In [VG84], Henver and Yao extended these algorithms to Algorithm G that processes general distributed queries. Apers [MP91] showed that this algorithm had some serious drawbacks. The analysis of the quality of the derived processing strategies is difficult. Both Henver [P85] and Apers [WE77] recognized these problems and developed the improved algorithm GENERAL. It is more clearly understood and more usable as a distributed query optimization algorithm than Algorithm G.

Generally Speaking, AHY algorithm has three steps:

1. Do all initial local processing.

   Local processing includes the computation of restrictions, projections, and semi-joins between relations that reside in the same node.
2. Decompose the query into simple queries.
3. Integrate the decomposed queries from step 2 into a near optimal execution strategy.

There are two primary versions of Algorithm GENERAL. To minimize response time of a processing strategy, using Algorithm PARALLEL and procedure RESPONSE emphasizes parallel data transmissions. To minimize the total time of a processing strategy, serial time transmissions are emphasized by the use of algorithm SERIAL and procedure TOTAL in algorithm GENERAL.

Since there exists some identical data transmissions, which leads to increase in total time, the third version of Algorithm GENERAL (corrective) is developed. Therefore, the redundancy problems can be solved.

Algorithm GENERAL can be applied to any general distributed query environment. It is relatively simple to program and has added the flexibility that all versions can be implemented.

2.7 Composite Semijoin

A composite semijoin is a semijoin in which the projection and transmission involve multiple columns. In most of the algorithms multiple semijoins may be performed in common source and common result sites. In this situation, it may be beneficial to do the semijoin as one composite rather than as multiple single column semijoins. The idea of using composite semijoins is mentioned also in [AY79, CGS79].

Now let's look at one example of the composite semijoin. Suppose we have two relations R1 and R2. R1 has three attributes X, Y, and M. R2 also has three attributes X, Y, and N. Therefore, the join attributes are X and Y. The projection of join attribute X is value 1, 2, and 3. The projection of join attribute Y is value aa, bb and cc.

R1

X	Y	M
1	aa	434
1	bb	54
2	cc	34
3	cc	12

R2

X	Y	N
1	cc	5
1	aa	3
2	bb	4
3	bb	9

X	Y	N
1	aa	3

FIGURE 2.16 COMPOSITE SEMIJOIN

If we use one single join attribute of relation R1 which is either X or Y to reduce relation R2, then no tuples will be reduced. However, there will be significant reduction when a composite semijoin is used. Using projection of two join attributes X and Y to reduce the relation R2, then only one tuple will be left.

R1

X	Y	M
1	aa	434
1	bb	54
2	cc	34
3	cc	12

R2

X	Y	N
1	aa	4
1	bb	5
2	cc	33
3	cc	143

FIGURE 2.17 COMPOSITE SEMIJOIN II

Here is another example. We have two relations R1 and R2. The join attributes of relation R1 and R2 are X and Y. If we use the composite semijoin instead of semijoin, no tuples will be reduced in relation R2, and meanwhile the cost is increased since 8 units have been transferred. Therefore, the composite semijoin doesn't have the benefit all the time; it depends on the situation and how you use it.

2.8 Dynamic Query Processing

More researchers turn their attention to multi-query optimizations [AHY78b, G84, H79, ML86, MOL00, S88, ST89, W85, YCB+86] and dynamic query processing [ACV+84, CE86, IB86, L83, Y86]. Furthermore, most researchers concentrate on the important class of select-project-join queries [H79]. They assume a single query environment and static processing. In a single query environment, performance of a single query is optimized, while static processing implies that an optimized strategy is not modified (it remains static) once its execution commences.

2.8.1 Heuristic Algorithm

In order to find an optimizing algorithm to formulate/form a strategy, which is a sequence of relational operations and locations for their execution to process a given query, Heuristic algorithm was proposed [CY93].

<u>Algorithm for a complete space of strategies</u> [CY93]

```
Input:  Query q
        Selectivities
        Statistical information (about the relations)
        Network locations of relations
        Delay (due to CPU processing and network data transfer unit of data)
Output: Space of strategies
```

Procedure:

Step 1: Create elements of level 0 – relations referred to by the query

Calculate their cost (delay) and size (the execution of restriction and projections)

Remove attributes (which are neither target nor joining)

Step 2: do step 3 for p =1, 2,3…m

Step 3: Insert elements representing results of p join executions into the level p of the graph of strategies by executing step 4 for $p_X = p-1, p-2 \ldots, \lfloor p/2 \rfloor$

Step 4: Let the graph level py be such that $p=1+px+py$ $(px \geq 0, py \geq 0)$

Representing a relation R_i of each element of the level px

Search the level py for elements which can be joined with R_i , and representing each such element a relation R_j

Create z new elements,each representing the result of the join of R_i with R_j executed at one of the z unique network locations

Calculate their size and delays and insert them in the level p of the graph of the strategies and save the information

Example: We have three relations represented by tree. Nodes stand for relations, and edges stand for join attributes.

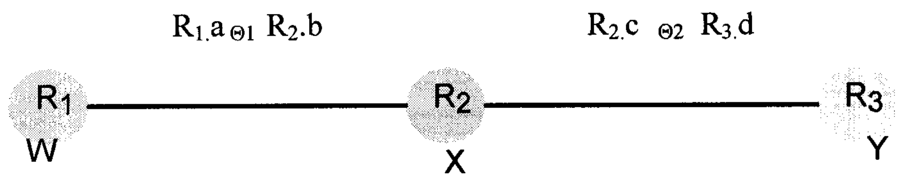

FIGURE 2.18 HEURISTICS ALGORITHM

Relation R1 has join attribute with R2, and relation R2 has join attribute with R3. In this algorithm, we don't care about what the join attributes are. Note, these relations are located in different sites, W, X, and Y respectively.

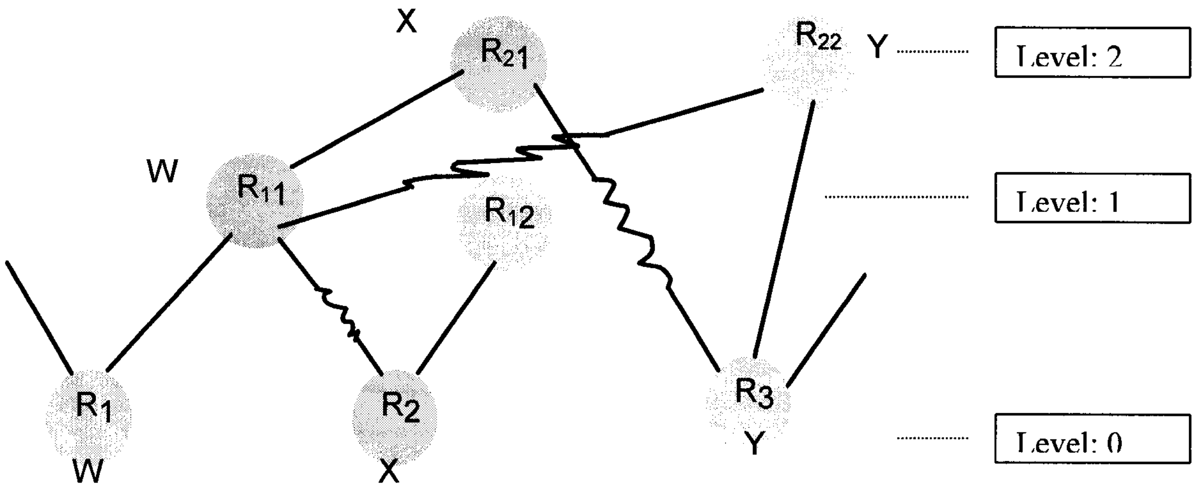

FIGURE 2.19 HEURISTICS ALGORITHMII

First, we list all three relations in level 0, so in this case we have three nodes. Next, build level 1 by combining all the possible joins. Therefore, we have 6 combinations: R1 JOIN R2 and results were sent to R1 W; R1 JOIN R2 and results were sent to R2 X; R1 JOIN R3 and results were sent to R1 W; R1 JOIN R3 and results were sent to R3 Y; R2 JOIN R3 and results were sent to R2 X; R2 JOIN R3 and results were sent to R3 Y. Continue till level m is completed. (You can pick any number of m) Finally, after calculating the cost, we will pick one of these paths, which is the optimal solution.

<u>Heuristic algorithm</u>

This algorithm is a modification of OPT_GV, which will decide the sequence and network locations for executing joins [BR88a]. In this algorithm, one more input N is used. In the above example we can see that OPT_GV will be infinitely and exponentially increased. If say, we decide to stop at level N, and then we will pick the best solution in level N, and remove all the unnecessary join nodes in and below level N. Therefore, we will have less computation and cost when moving up to N+1 level. So, N becomes quite important. An increase in the value of N results in an increase in overhead, but the execution delay of strategies would be lower.

2.8.2 AJL (Abort Join Last) Method

Normally, after a query is parsed into a canonical form, an optimizing algorithm formulates a query processing strategy (QPS), which specifies the sequence in which relational operations are executed and the network locations of their execution [BS92]. In QPS formulation, estimating techniques are used to determine the size of intermediate results [BLM89, C84, CE86, CY93, ES80, IB86, ST89, W82]. In Adaptive approach, execution strategy is monitored and corrections are made if the estimation of the intermediate result is inaccurate.

Within the field of adaptive query processing there remains the important question of when to correct a strategy. Two general methods have been proposed. The first is reformulation, in which the unexecuted portion of the strategy is reformulated at every intermediate stage on the basis of available updated information. If the new strategy is estimated to reduce cost, then correction is appropriate. The second method is the threshold method, in which reformulation occurs only when intermediate results exceed a predetermined threshold or lie outside a specified band of values. [BR88b] refines this approach by the use of one threshold value per intermediate result. [BR88a] addresses both the monitoring and execution phases of strategy execution. AJL method is another alternative way, which can decide the correction [BS92].

<u>Three phases for adaptive processing [BS92]</u>

1. Initial (static) QPS formulation
2. Strategy execution
3. Strategy correction

Alternative Strategy Example:

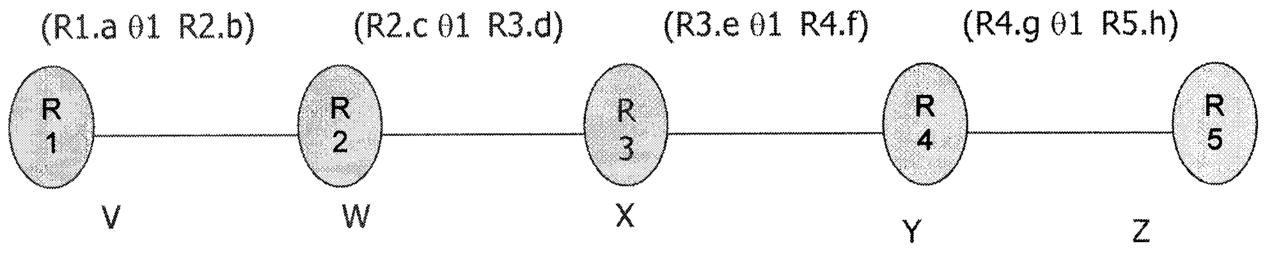

FIGURE 2.20 AJL RELATIONS

We have 5 relations in different sites. It will be presented by tree structure.

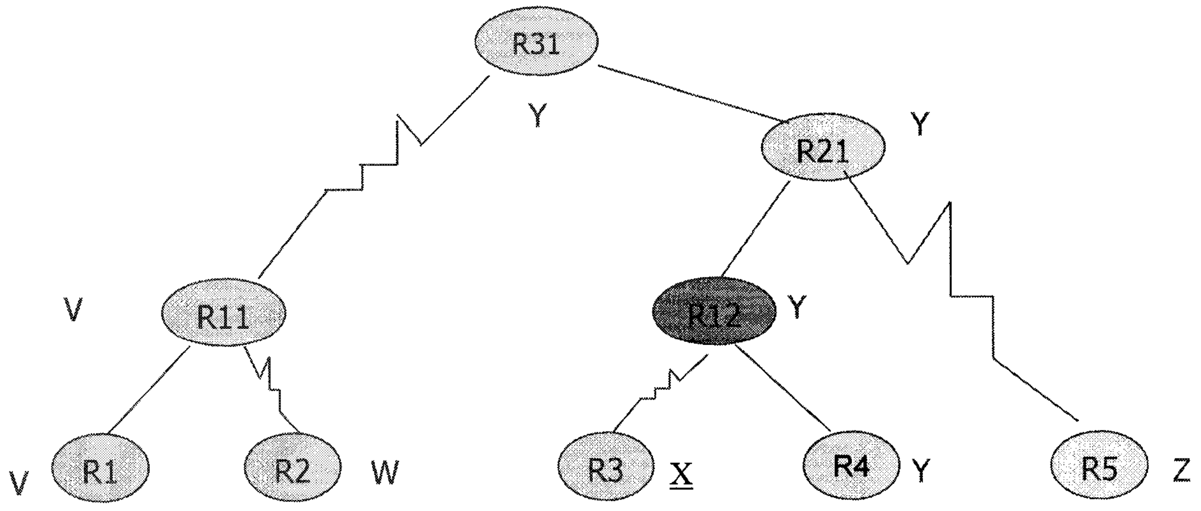

FIGURE 2.21 STRATEGY PRESENTED BY A TREE

The join procedure is shown in Figure 2.21 R1 JOIN R2, R3 JOIN R4 as R12 stored in site Y, R12 JOIN R5 stored in Y, and R11 JOIN R12 as R31 stored in Y. The alternative strategy instead of join R3 and R4 presented in Figure 2.2 is R11 JOIN R3 stored in V, and R4 JOIN R5 stored in Y (Figure 2.22).

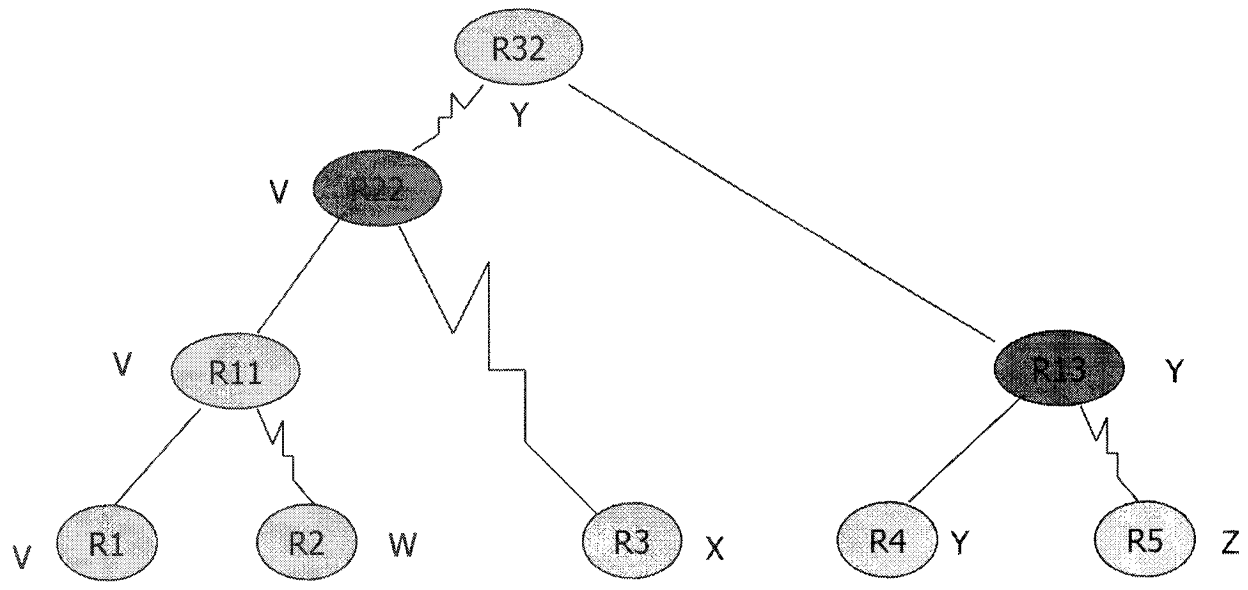

FIGURE 2.22 ALTERNATIVE STRATEGY

Now using AJL method, we can decide when and whether we shall use the alternative strategy. Here is the procedure:

1. Given a query, a formulator 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 the join execution, sampling methods are 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 and the current strategy will be aborted and the alternative strategy is used. Otherwise, the original strategy is allowed to continue.

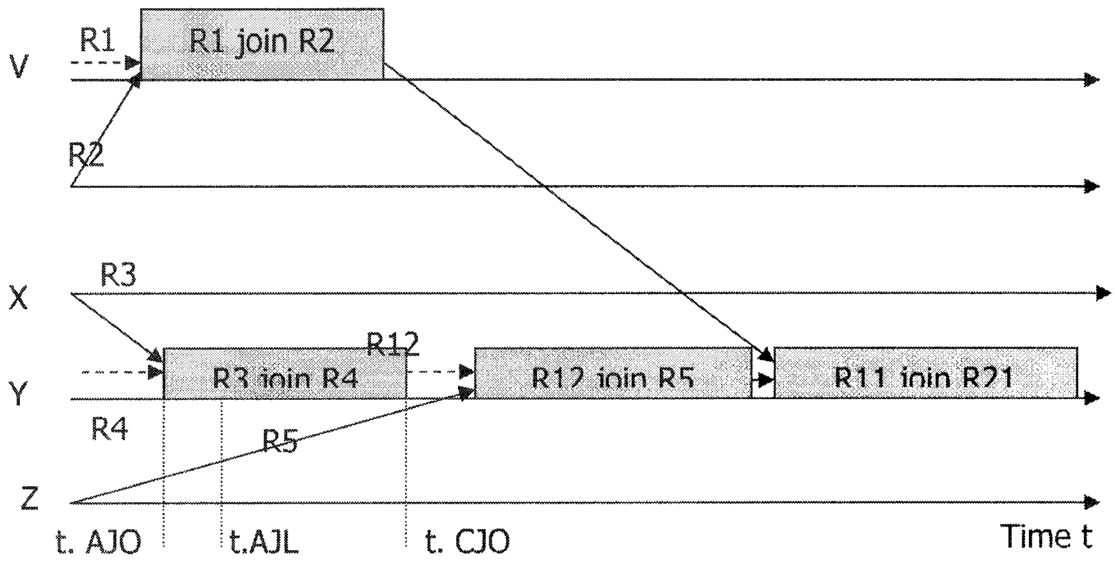

FIGURE 2.23 STRATEGY TIMING DIAGRAM

So, in this example, during the original execution of R3 join R4, the alternative strategy will be formed. We will use the sampling data to estimate the total cost and compare it with the alternative strategy. That means, at 10/100 execution of R3 JOIN R4, we will use 10/100 data to estimate the total data. If the alternative strategy is better than the original one, then we will abort the original one. We can also choose 20/100 or any other number instead of 10/100.

We can see that the fraction of the join used by the sampling method to determine the size of the join result is quite important. If we can estimate just before the execution which is 0/100, t. AJO (Abort Join Optimal), the sampling cost will be eliminated. However, it's not realizable and we have to assume we have perfect priori knowledge of the intermediate result sizes. If we use CJO (Complete join optimal), which is 100/100, only the join is completed is a new strategy formulated and instituted for the remaining unprocessed portion of the strategy, then the sampling cost will be increased while the estimation is more accurate.

2.9 Filter Based Algorithm

The difficulty of the optimization of the general queries in a distributed database system is the sequence and location of the operation. Traditionally, join or semijoin based algorithms are used to get the optimal solution. Compared with that, the cost of constructing the filter is much less. Moreover, it's cheaper to transfer a filter over the network than a relation. The disadvantage of using a filter is the collisions which occur as a result of two or more attributes values hashing to the same address in the array.

2.9.1 Bloom Filter Based Algorithm

The term "Bloom filter" comes from Bloom [LT95]. By definition [SL83], a Bloom filter is an array of bits which functions as a very compact representation of the value of a join attribute [JM93]. The steps for constructing a Bloom filter are:

1. Construct an array and set all bits to zero.
2. For each value of the join-attribute use hashing.
3. For each address produced, set the corresponding bit into the array to one.

Obviously, using Bloom filter, the transfer cost will be greatly reduced. However, the problem of collisions occurs. In order to solve this problem, one big Bloom filter can be split into a number of small Bloom filters [B70, BR88]. This also leads an efficient usage of main memory [M90].

Given two relations Ri and Rj with join attribute A. The steps for this Algorithm are [TC92]:

1. Calculate the bit array size and initialize to zero.
2. For each join attribute value in $R_i[A]$, use a hash semijoin described in [CCY92] to produce d address in the bit array. Set the corresponding bit to 1.
3. Ship the bit array to the site of R_j.
4. For each join attribute value in $R_j[A]$, produced addresses in the bit array using the same d hash function. If d address in the array is set to 1, then the tuple kept, otherwise, reject the tuple.

Example:

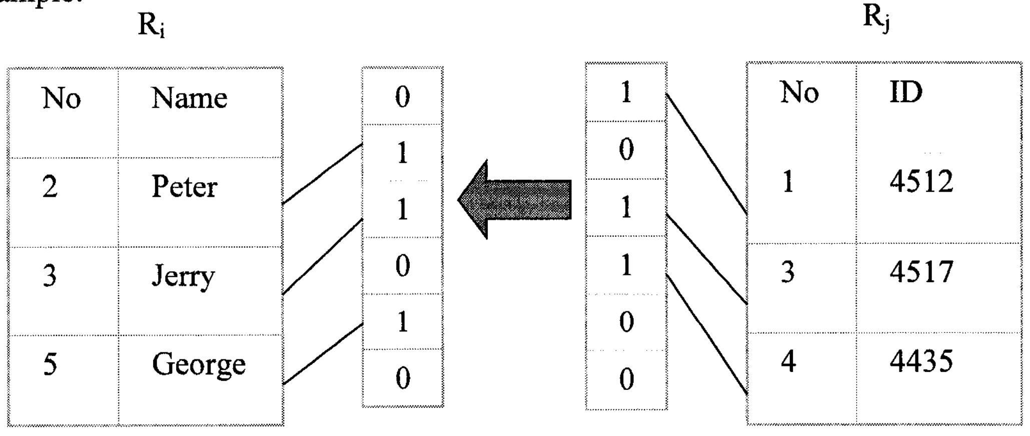

FIGURE 2.24 BLOOM FILTER OPERATION

There are two relations, R_i and R_j. The join attribute is No. First we create array of bits, and initialize it to zero. Since the value in attribute No of relation R_j is 1, 3, and 4. We set the related bits to 1, and ship the array to site i. We produce one array in site i and initialize it to zero. Since the values of join attribute No is 2, 3, and 5. We set these bits to 1. Finally, we compare two arrays of bits. If the value does not match, then tuple will be rejected. Therefore in this example only one tuple with No 3 will remain. All the rest will be removed.

Chapter 3

Algorithm H

In this chapter, a new algorithm is presented and detailed example also indicates the processing steps.

3.1 Motivation

The pipeline N-way join algorithm eliminates intermediate results and reduces relations access cost. However, its join sequence is sequential, and data transfer cost is increased. A new algorithm which applies the pipeline idea to a filter based algorithm is proposed. This algorithm is the combination of pipeline N-way join algorithm and an algorithm developed in [JM03]. The main goal of this new algorithm is to reduce data transfer cost while maintain low I/O cost as in pipeline N-way join algorithm.

3.2 Description of the Algorithm

This algorithm uses reduction filters. Each filter is an array of bits that functions as a very compact representation of the value of a join attribute in a relation. A "perfect" hash function is used to set bits in the filter.

This algorithm can process general queries consisting of an arbitrary number of relations and join attributes. Each query is represented by a graph and an adjacency list. Filter based pipeline N-way join algorithm consists of three phases: forward reduction, backward reduction and query processing phase.

Phase One

1. Select the relations with the lowest in-degree (number of join attributes) for processing.
2. Build Tuple Connector (C_i) in local site by using existing filters.
3. Construct filters for all join attributes. If a filter is already available then update the values.
4. Use adjacency list to "remove edges" from query graph, that is, reduce the in-degree of each relation in the list by 1.
5. Mark relation as processed and put it in queue.

Repeat all steps until all relations have been processed once.

Phase Two

1. Get relation R_i from queue, construct tuple connector C_i'.

2. Build pipeline cache planner (PCP_i).

Repeat all steps until the queue is empty.

Phase Three

1. Send Pipeline Cache Planner to query site

In order to understand the algorithm, we will use the same example as N-way pipeline join algorithm. We have 3 different relations in 3 different locations. @R is tuple ID.

R1

@R1	A	B	C	E
@R11	**5**	**4**	3	6
@R12	**1**	**4**	2	4
@R13	**3**	**3**	2	3

R2

@R2	A	D
@R21	**1**	4
@R22	**5**	6
@R23	7	5

R3

@R3	B	C	F
@R31	**3**	**2**	9
@R32	**5**	**2**	6
@R33	**6**	7	5
@R34	**4**	**3**	4

FIGURE 3.1 ALGORITHM H EXAMPLE

In order to get join attributes, we build Matrix first. Initially, we set all the bits to 0. Then we check the attributes in Relation R1. Since R1 has attributes A, B, C, and E. We set bits from 0 to 1. Same as R2 and R3, we set bits to 1.

	A	**B**	**C**	**D**	**E**	**F**
R1	1	1	1	0	1	0
R2	1	0	0	1	0	0
R3	0	1	1	0	0	1

FIGURE 3.2 MATRIX

From the above figure, we can see that more than one relation have the same attributes A, B, and C. And only one relation contains attribute D, E, and F. Therefore, we can conclude that A, B, C are join attributes.

Next, we build adjacency list.

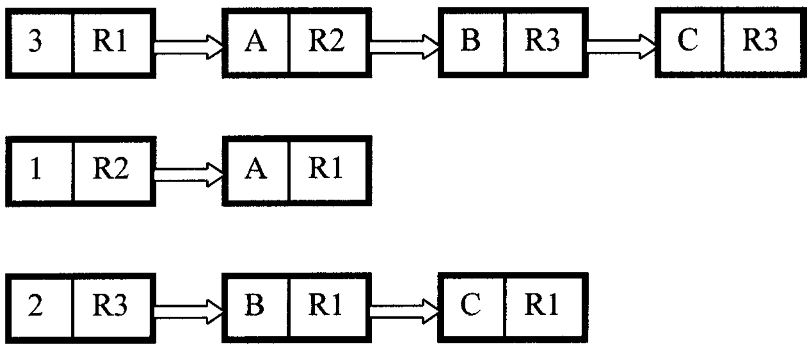

FIGURE 3.3 A-LIST

R1 has three join attributes A, B, and C. It has the same join Attribute A with R2, and has the same join attributes B, and C with R3. So, we put 3 in its head node and link join attributes to R2 and R3. R2 has one join attribute A. Therefore we set 1 in head node and link it to R1 which has the same join attribute A. Finally, we check R3, and it has two join attributes B, and C. So, we put 2 in its head node and link it to R1.

- **Phase One**

<u>Step 1.1</u>

According to the algorithm, we start phase one first, which is forward reduction. We scan the A-list, and we find that 1 is the smallest number in head node. So, we process R2 first. Since R2 is the first relation to be processed, C2 will be the projection of R2' join attributes.

@R2	A
@R21	1
@R22	5
@R23	7

FIGURE 3.4 C2 CONNECTOR

Step 1.2

We construct filters for join attributes A and get:

Filter A: 1, 5, 7

Step 1.3

Use adjacency list to "remove edges" from query graph, that is, reduce the in-degree of each relation in the list by 1. Mark relation as processed and put it in queue. Since R1 is linked to R2, head node will be reduced from 3 to 2. R2 is processed, so it is marked as

-1.

2	R1
-1	R2
2	R3

FIGURE 3.5 HEAD NODES

Step 2.1

Scan the A-list, and we find that 2 is the smallest number in head node. So, we process R1 and make C1 in local site. We project C2 connector' join attributes, join it with R1 and get C1 Connector:

@R1	A	B	C
@R11	5	4	3
@R12	1	4	2

FIGURE 3.6 C1 CONNECTOR

<u>Cost</u>

Since projection of C2 connector's join attribute, which is the value of 1, 5, and 7 are forwarded to R1. The cost is 3 units.

<u>Step 2.2</u>

Update filter A, and construct filters B and Ct:

Filter A: 1, 5

Filter B: 4

Filter C: 2, 3

<u>Step 2.3</u>

Since R1 is linked to R3, head node will be reduced from 2 to 1. R1 is processed, so it is marked as -1.

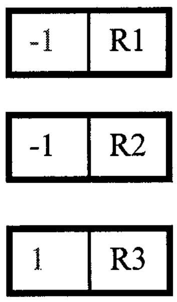

FIGURE 3.7 HEAD NODES

<u>Step 3.1</u>

Scan the A-list, and we find that 1 is the smallest number in head node. So, we process R3 and make C3 in local site. We project C1 connector' join attributes, join it with R3 and get C3 Connector:

@R3	A	B	C
@R34	5	4	3

FIGURE 3.8 C3 CONNECTOR

Cost

Since projection of C1 connector's join attribute, which has 6 units are forwarded to R3, The cost is 6 units.

Step 3.2

Update filter A, B and C:

Filter A: 5

Filter B: 4

Filter C: 3

Step 3.3

R3 is processed, so it is marked as -1.

-1	R1

-1	R2

-1	R3

FIGURE 3.9 HEAD NODES

Since all the relations have been processed, we will move to phase two. The total cost in phase one is 3+6=9 units.

- **Phase Two**

Step 1.1

Get relation from queue which is R3, and construct tuple connector C3'. Since it is the first relation in queue. C3' connector is same as C3.

@R3	A	B	C
@R34	5	4	3

FIGURE 3.10 C3' CONNECTOR

Step 1.2

Construct Pipeline Cache Planner in site R3.

@R3
@R34

FIGURE 3.11 PCP3

Step 2.1

Get relation from queue which is R1, and construct tuple connector C1'. C1' is the join of C1 and projection of C3's attributes. C1 Connector has two tuples and C3 Connector has one tuple. Comparing the value of join attributes, only one tuple will be included in C1' Connector.

@R3	A	B	C
@R34	5	4	3

FIGURE 3.12 C1' CONNECTOR

Cost

Since the projection of C3' connector's attributes is forwarded to C1, the cost is 4 (3 units plus 1 unit of tuple ID).

Step 2.2

Construct Pipeline Cache Planner in site R1.

@R1	@R3
@R11	@R34

FIGURE 3.13 PCP1

Step 3.1

Get relation from queue which is R2, and construct tuple connector C2'. C2' is the join of C2 and projection of C1's attributes.

@R1	A	B	C
@R11	5	4	3

FIGURE 3.14 C2' CONNECTOR

Cost

Since the projection of C1's attributes is forwarded to C2, the cost is 4 (3 units plus 1 unit of tuple ID).

Step 3.2

Construct Pipeline Cache Planner in site R2.

@R1	@R2	@R3
@R11	@R22	@R34

FIGURE 3.15 PCP2

Since all the relations have been processed in queue, we will move to phase three.

The total cost in phase one is 4+4=8 units.

- **Phase Three**

We send Pipeline cache planner to query site. In this case, the total cost is 2+4+6=12 units.

The total cost in this example is 9 + 8 + 12 = 29 units.

Now, let's check if the tuple ID in PCP is correct or not. In PCP2, it has one tuple with value of @R11, @R22, and @R34.

R1

@R1	A	B	C	E
@R11	5	4	3	6

R2

@R2	A	D
@R22	5	6

R3

@R3	B	C	F
@R34	4	3	4

FIGURE 3.16 REDUCED RELATIONS

If ship all the relations R1, R2, and R3 to query site and join them there. We will get the same result as shown in figure 3.16. Therefore, we can say we can get the same results as we choose IFS.

Chapter 4

Experiments and Evaluation

In order to confirm whether algorithm H has better performance than the Pipeline N-way join algorithm and IFS, I implemented Pipeline N-way join algorithm and algorithm H. Various experiments based on a large number of queries were made also. In this chapter, we detail the experimental scenario and summarize results.

4.1 Assumptions

The proposed algorithm H is based on the following assumptions:

1. A distributed relational database management system is connected via a point-to-point network.

2. The relations are distributed amongst the nodes and all nodes can access all data. Each relation has at least one join attribute.
3. Only selected-projection-join (SPJ) queries are considered. No other operations such as UNION, INTERSECTION, and DIFFERENCE are used.
4. A perfect hash function is assumed when building filters.
5. Each attribute is a 64-bit word
6. The cost model is:

   $C(X) = C_0 + X$

   For simplicity, $C_0= 0$. X is the number of data transmitted. We used “word” as the unit.

4.2 Methodology

The framework for evaluating algorithm H is based on the comparison with Pipeline N-way join algorithm and IFS (Initial Feasible Solution). IFS is one algorithm which ships all the relations to query site and performs Join operations there.

4.2.1 Experimental System

The experimental system includes six parts. They are Database related statistical information generator, relation generator, IFS, pipeline N-way join algorithm, algorithm H, and analysis program. Both pipeline N-way join algorithm and algorithm H were fully implemented by me.

Todd Bealor [TB95] member of the Database Research Group in University of Windsor programmed statistical information generator and relation generator. They will be described in detail in 4.2.2. IFS will be introduced in 4.2.3. Pipeline N-way join algorithm was discussed in 2.5, and algorithm H was discussed in chapter 3. In 4.2.4, we will describe analysis program in detail.

4.2.2 Statistical Information Generator [TB95]

Database Research Group programmed statistical Information Generator. It has the following characteristics:

- Relations and Attributes

  Each query consists of 3 to 6 relations, and 2 to 4 join attributes. Therefore, there are twelve combinations.

- Number of tuples in a relation

  Each relation has between 150 and 2000 tuples.

- Attribute Domain Size

  Domain is the set of allowable values for the attribute.

  The size of each join attribute domain is in the range between 100 and 600.

- Selectivity

  Selectivity is the ratio of distinct attribute values over the attribute domain size. It shows the reduction power of projection. If the ratio is low, then it has high selectivity and reduction power. Vice versa, if the ratio is high, then it has low selectivity and reduction power. For example, the ratio 0.1 has higher selectivity and more reduction power than that of ratio 0.9. In this experiment, the ratio is between 0.1 and 0.9.

All the above are implemented by program create_query.c and relbuilder.c. They were written in language C. Next, we describe these two programs in more detail.

- <u>create_query. c</u>

  This program generates statistics information. The inputs are the number of relations and join attributes. For example, create_query 3 2 generates three relations and two join attributes. The output has three parts. They are dbstats, domains and summarize information for each relation.

  (1) dbstats

    This file contains the number of relations and joining attributes. It also shows relation cardinality, attribute cardinality and selectivity.

  (2) domains

    This files shows domain size for each join attribute.

  (3) Statistics information for each relation

    Some files will be generated depends on the relation numbers. If there are two relations, then two output files will be produced. Each files contains relation cardinality, number of join attributes, and for each join attribute, it shows attribute label, size of attribute and its domain.

Now, let's see one example. Execute create_query 4 2, it generates:

dbstats

```
4 2
1400    0 0.000000   580 0.644444
3500  790 0.840426   480 0.533333
900     0 0.000000   670 0.744444
4400  510 0.542553     0 0.000000
```

domains

```
940
900
```

Rel0

```
1400 1 1   580   900
```

Rel1

```
3500 2 0   790   940 1   480   900
```

Rel2

```
900 1 1   670   900
```

Rel3

```
4400 1 0   510   940
```

From the file dbstats, we can know that it has 4 relations and 2 join attributes. The first relation only has one join attribute. For relation 1, 1400 is relation cardinality, 580 is attribute cardinality and 0.644444 is the selectivity. For relation 2, 3500 is relation cardinality, 790 is the first attribute's cardinality and 0.840426 is its selectivity; 480 is the second attribute's cardinality and 0.533333 is its selectivity. Line 4 and line 5 indicates Relation 3 and 4's information.

In file domains, 940 and 900 are domain size for each join attribute.

Finally, it generates statistics information for each relation. In this example, since there are four relations, four files will be generated. Rel0 includes relation cardinality 1400. The first 1 means it has one join attribute, and second 1 is the attribute label. 580 is the size of join attribute and 900 is its domain. Rel1, Rel2, and Rel3 are the same.

- <u>relbuilder. c</u>

  Based on the output of create_query.c, relbuilder.c creates real relation. Input for relbuiler.c is a relation number. The output file starts with "R".

In the above example, since create_query 4 2 generates four relations, relbuilder 3, relbuilder 2, relbuilder 1, and relbuilder 0 will be executed. As a result, real relation R3, R2, R1, and R0 will be generated. Here is some data from R0:

```
R0
1
1
0  338
1  306
2  548
```

3 486

4 234

...

In first line, number 1 means it has one join attribute. In second line, number 1 is the label of attribute. From line three, it shows tuple number and real data.

4.2.3 Initial Feasible Solution (IFS)

Initial feasible solution is an algorithm which ship all the relation to query site directly. In order to evaluate the pipeline N-way join algorithm and algorithm H, we develop IFS program and compare cost with proposed method.

To be comparable, we use same example illustrated in pipeline N-way join algorithm and algorithm H.

R1

@R1	A	B	C	E
@R11	**5**	**4**	3	6
@R12	**1**	**4**	2	4
@R13	**3**	**3**	2	3

R2

@R2	A	D
@R21	**1**	4
@R22	**5**	6
@R23	7	5

R3

@R3	**B**	C	F
@R31	**3**	**2**	9
@R32	**5**	**2**	6
@R33	**6**	**7**	5
@R34	**4**	**3**	4

FIGURE 4.1 IFS EXAMPLE

There are three relations and sit in different sites. R1 has 16 units, R2 has 8 units, and R3 has 12 units. Therefore, total transmission cost for algorithm IFS is 16+8+12=36 units.

4.2.4 Analysis Program

In order to evaluate algorithm H, we construct an analysis program which shows the improved percentage.

Here is the evaluation method.

IFS vs. Pipeline N-way join algorithm:

$$\frac{C_{IFS} - C_{\text{pipeline N-way join algorithm}}}{C_{IFS}} * 100\% = \text{percentage improved}$$

IFS vs. algorithm H:

$$\frac{C_{IFS} - C_{\text{algorithm H}}}{C_{IFS}} * 100\% = \text{percentage improved}$$

Pipelines N-way join algorithm vs. algorithm H

$$\frac{C_{\text{pipeline N-way join algorithm}} - C_{\text{algorithm H}}}{C_{\text{pipeline N-way join algorithm}}} * 100\% = \text{percentage improved}$$

4.3 Results and Evaluation

Since each query contains 3 to 6 relations and 2 to 4 join attributes, it has 12 combinations. To simplify the problems, for each algorithm with each combination, the average cost is based upon 100 queries.

4.3.1 Pipeline N-Way Join Algorithm vs. IFS

First, we compare the data transmission cost between Pipeline N-way join algorithm and IFS.

TYPE	C_{IFS}	$C_{N\text{-WAY JOIN ALGORITHM}}$	% IMPROVED
3-2	11608	11051	4.80%
4-2	13097	12047	8.02%
5-2	19244	16248	15.57%
6-2	22745	18535	18.51%
3-3	18108	17350	4.19%
4-3	23242	21078	9.31%
5-3	26745	23174	13.35%
6-3	31395	25976	17.26%
3-4	22424	18452	6.41%
4-4	26933	21392	7.98%
5-4	35223	26894	8.90%
6-4	39120	29542	9.33%

FIGURE 4.2 PIPELINE N-WAY JOIN ALGORITHM VS. IFS

The first column indicates query type. For each combination cost will be shown in second and third column. Based on analysis program, improved percentage is record in fourth column.

- For each type cost of IFS is higher than Pipeline N-Way join Algorithm.
- No matter what algorithm is used, with the same join attribute, if there are more relations, the cost will be higher. For example: $C_{6-2} > C_{5-2} > C_{4-2} > C_{3-2}$.
- No matter what algorithm is used, with the same relation number, if there are more join attributes, the cost will be higher. For example: $C_{3-4} > C_{3-3} > C_{3-2}$.
- The average improved percentage is around 10+%.

Here is the graph that shows cost.

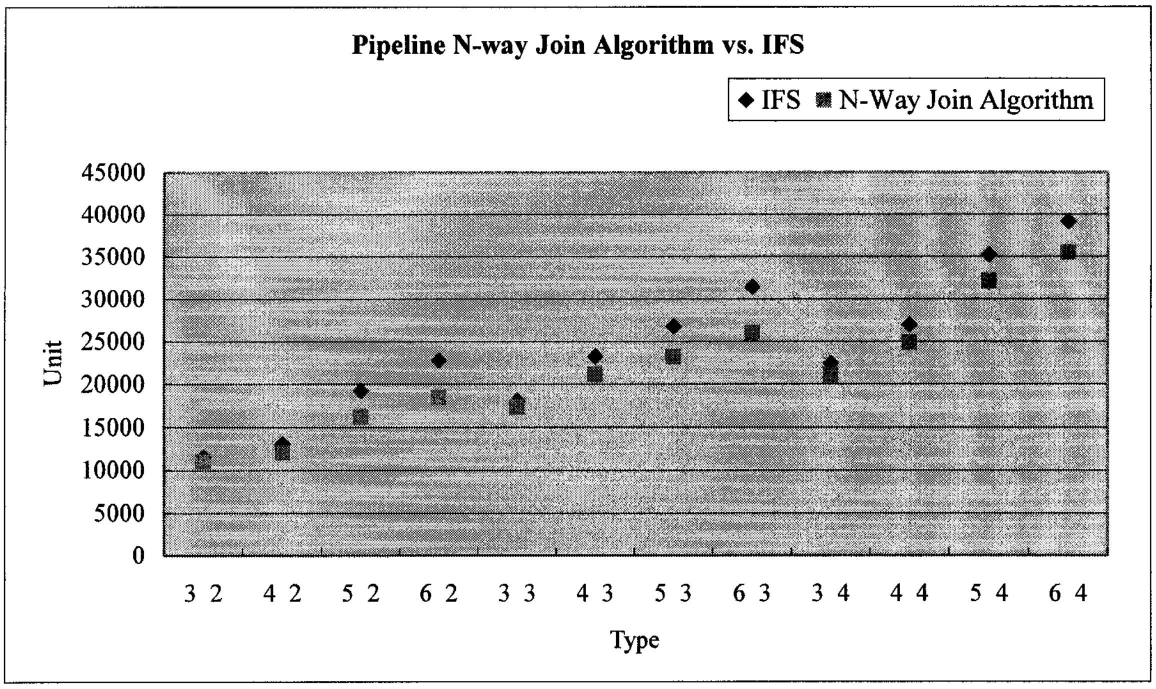

Figure 4.3 Pipeline N-way join Algorithm vs. IFS - Graphic

4.3.2 Algorithm H vs. IFS

Now, we compare the data transmission cost between IFS and algorithm H.

TYPE	C_{IFS}	$C_{ALGORITHM\ H}$	% IMPROVED
3-2	11608	9842	15.21%
4-2	13097	10542	19.51%
5-2	19244	14025	27.12%
6-2	22745	15754	30.74%
3-3	18108	15348	15.24%
4-3	23242	18349	21.05%
5-3	26745	20129	24.74%
6-3	31395	22145	29.46%
3-4	22424	18452	17.71%
4-4	26933	21392	20.57%
5-4	35223	26894	23.65%
6-4	39120	29542	24.48%

FIGURE 4.4 ALGORITHM H VS. IFS

From the above figure, we can see that:

- For each type cost of IFS is higher than Algorithm H.
- No matter what algorithm is used, with the same join attribute, if there are more relations, the cost will be higher. For example: $C_{6-3} > C_{5-3} > C_{4-3} > C_{3-3.}$
- No matter what algorithm is used, with the same relation number, if there are more join attributes, the cost will be higher. For example: $C_{4-4} > C_{4-3} > C_{4-2.}$
- The average improved percentage is around 20+%.

Here is the graph that shows cost.

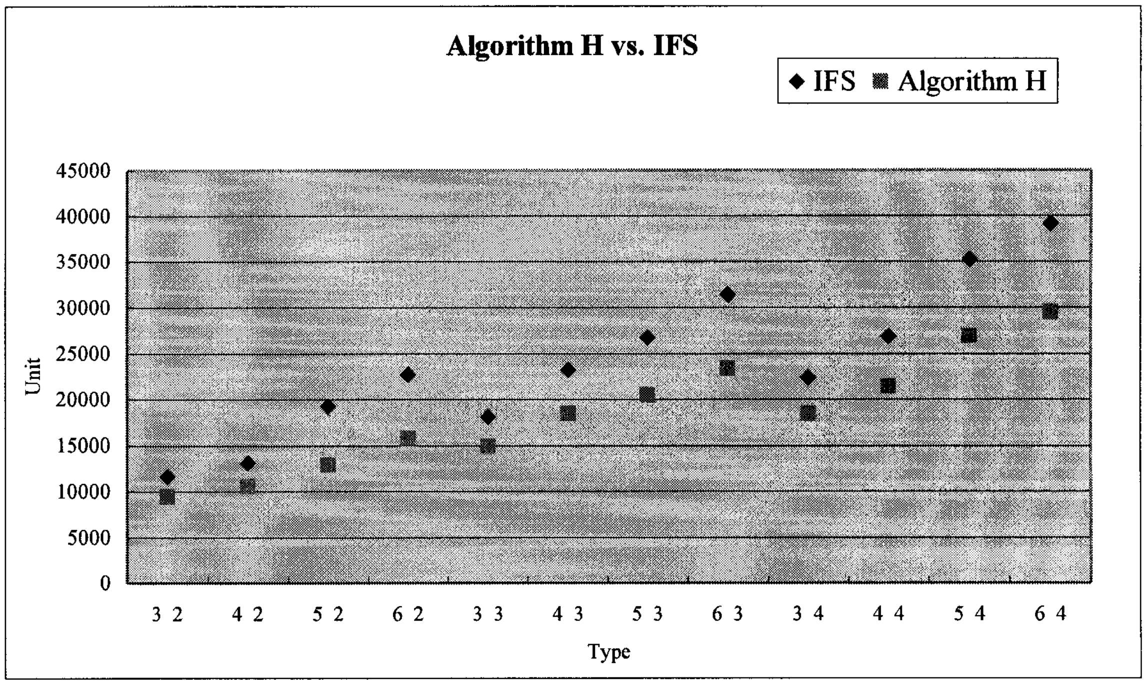

FIGURE 4.5 ALGORITHM H VS. IFS - GRAPHIC

4.3.3 Algorithm H vs. Pipeline N-way join algorithm

Finally, we compare the data transmission cost between algorithm H and Pipeline N-way join algorithm.

TYPE	$C_{\text{N-WAY JOIN ALGORITHM}}$	C_H	% IMPROVED
3-2	11051	9842	10.94%
4-2	12047	10542	12.49%
5-2	16248	14025	13.68%
6-2	18535	15754	15.00%
3-3	17350	15348	11.54%
4-3	21078	18349	12.95%
5-3	23174	20129	13.14%
6-3	25976	22145	14.75%
3-4	18452	18452	12.08%
4-4	21392	21392	13.69%
5-4	26894	26894	16.19%
6-4	29542	29542	16.72%

FIGURE 4.6 ALGORITHM H VS. PIPELINE N-WAY JOIN ALGORITHM

From the figure, we can get that:

- For each type cost of Pipeline N-way join algorithm is higher than Algorithm H.
- No matter what algorithm is used, with the same join attribute, if there are more relations, the cost will be higher. For example: $C_{6\text{-}2} > C_{5\text{-}2} > C_{4\text{-}2} > C_{3\text{-}2}$.
- No matter what algorithm is used, with the same relation number, if there are more join attributes, the cost will be higher. For example: $C_{3\text{-}4} > C_{3\text{-}3} > C_{3\text{-}2}$.
- The average improved percentage is around 10+%.

Here is the graph that shows cost.

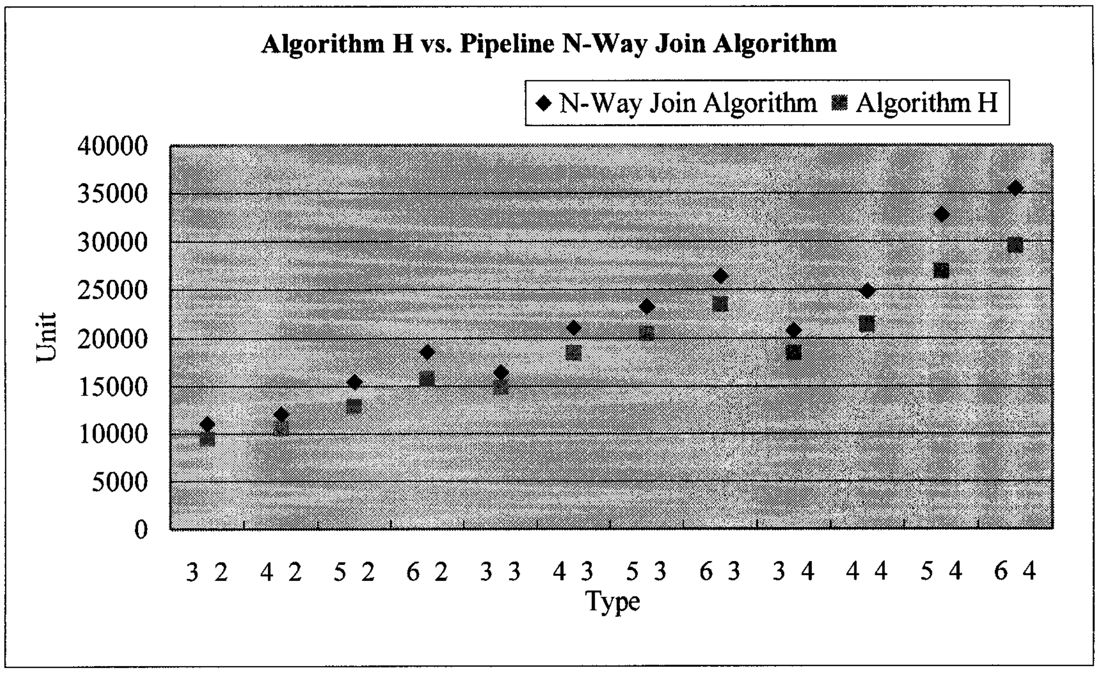

FIGURE 4.7 ALGORITHM H VS. PIPELINE N-WAY JOIN ALGORITHM - GRAPHIC

4.4 Evaluation

After doing a large number of queries, we can get the conclusion that algorithm H is has the lowest transmission cost compares with the other two.

Both IFS and Pipeline N-way join algorithm ship relations to other sites. However, they have two differences. In IFS relations ship to query site without being processed, while in Pipeline N-way join algorithm projection of join attributes will be sent to query site. Besides, in IFS shipping relations is one way, and in Pipeline N-way join algorithm it has forward and backward transmission. Obviously, pipeline N-way join algorithm improved Initial Feasible Solution, however it's only 10+%. The reason why the improved percentage is not that high is pipeline N-way join algorithm has two-way reduction, therefore it sends part of the original relations once and temporary relations (Tuple Connectors).

Algorithm H applies pipeline idea to a filter based algorithm. Both Algorithm H and Pipeline N-way join algorithm have three phases, and there are no intermediate results generated. The main difference is in Algorithm H relations are not processed sequentially. The other difference is Algorithm H uses filer concept, therefore transmission cost is lower than using projection of join attributes. Since the experiment only carries 4 to 6 relations and 2 to 4 join attributes. The sequence of process relations maybe same in both algorithm H and pipeline N-way join algorithm. Therefore, the advantage of using algorithm H is not that obvious.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Query optimization is one of the major research areas in distributed database systems. The problem is to find a strategy for executing each query over the network in the most cost-effective way.

In the last two decades, a variety of algorithms has been studied to improve performance. In this thesis, we have discussed some of the main algorithms, such as Join, Semijoin, two-way semijoin, filter based join algorithm. Based on past research, a new algorithm, filter based pipeline N-way join algorithm, is presented to reduced data transfer cost.

Algorithm H keeps the advantages that pipeline N-way join algorithm has. During the query processing, no intermediate results have been generated. All the relations are only accessed once, therefore, it has low I/O cost. Instead of sending projection of join attributes, Algorithm H makes use of filter concept and it creates filter for each join attribute.

From a large number of experiments described in Chapter 5, we can make the conclusion that algorithm H has a better performance than that of IFS and Pipeline N-way join algorithm. Compare to IFS, it has 20+% improved. And it has 10+% improved percentage compared with Pipeline N-way join algorithm.

5.2 Future Work

Though algorithm H has lower data transmission cost than IFS and Pipeline N-way join algorithm, there are some small problems to be concerned.

First, to simplify the problem, we use perfect hash function in algorithm H. However, in real situation some other hash functions will be used which leads the problem of collision. Collisions always exist in filter-based algorithms; more research on this area should be studied.

Secondly, in our experiments, selectivity is in the range between 0.1 and 0.9. However, selectivity is also one fact to evaluate the algorithm, it shall be classified into three categories. For example, in the range between 0.1 and 0.3, we named it as high selectivity. In the range between 0.4 and 0.7, it's a middle selectivity. And in the range between 0.4 and 0.7, it's a low selectivity. In the future, we need to do more tests based on different selectivity range.

Reference

[A79] P.M.G. Apers, Critique on and improvement of Henver and Yao's distributed query processing algorithm G, Vrjie Univ., Amsterdam, The Netherland, IR 48, Feb. 1979

[ACV+84] L. P, Arbee, V Chen, P.K. Li, Improvement Algorithms for Semijoin Query Processing Programs in Distributed Database Systems, IEEE Trans on Comput., vol. C-33 1984

[AHY78a] P. Apers, A. R. Hevner, and S. B. Yao, Optimization of data access in distributed Systems. Computer Science Department. Tech. Rep. TR281, Purdue Univ. July 1978

[AHY78b] A. R. Henver and S. B. Yao, Query processing in distributed database systems, in Proc. 3rd Berkley Workshop on Distributed Data Management and Comput. Networks, Berkeley, CA, 1978

[AHY83] P. Apers, A. R. Hevner, and S. B. Yao, Optimization Algorithm for Distributed Queries, IEEE Trans. Software Engineer SE-Vol 9.Issue 1, Jan. 1983

[AR91] A. Amir and N. Rossopoulous. Optimal view caching, Inform. System Vol. 15. No. 2. pp 169-171, 1991

[AY79] A. R. Hevner, and S. B. Yao. Query Processing in Distributed Database System, IEEE Trans, on Software Engineer SE-Vol 5, Issue 3, May 1979

[B70] E. Babb, E. Implementing a relational database by means of specialized hardware. ACM Trans. Database System, Vol. 13, no. 7, July 1970, pages 422-426

[BGW+81] P.A. Berstein, N. Goodman,E. Wong, C. L. Reive, and J. Rothnie. Query processing in a system for distributed databases ACM Trans. Database Systems 6, No 4. Dec, 1981

[BLM89] D. A. Bell. D. H. O. Ling and S. McClean, Pragmatic estimation of join sizes and attribute correlations, in Proc Fifth Int. Conf. On Data Engineering, 76-84 1989

[BR88a] P. Bodorik, J. S. Riordon. Heuristic algorithms for distributed query processing, In Proceedings of the International Symposium on Databases in Parallel and Distributed Systems, pages 144-155, 1988

[BR88b] P. Bodorik, JS Riordon A Threshold mechanism for distributed processing of queries, In Proc. ACM CSC'88 Conf., Atlanta, GA, pp. 616-625 Feb. 1988

[BS92] P.Bodorik. J. Spruce. Deciding to Correct Distributed Query Processing, IEEE, June Vol. 4, No. 3. pp. 253-265 1992

[C82] T. Cheung. A method for equijoin queries in distributed relational databases. IEEE. Trans. Comput. C-31, No 8 Aug 1982

[C84] S. Christodoulakis, Implications of certain assumptions in databse performance evaluation, ACM TODS, vol. 9, no. 2, pp. 173-186, June 1984

[CBY84] D. M. Chui, P. A. Berstein, and Y. C. Ho, Optimizing chain queries in a distributed database system; SIAM J. Computer Vol. 13, no1, pp.116-134, Feb 1984

[CC83] C. T. Yu, and C. Chang, On the Design of a Query Processing Strategy in a Distributed Database Environment, Proc. ACM SIGMOD Int. 1983 pp30-39

[CCY92] T. Chen, A. L. P. Chen and W. Yang. Hash-semijoin: A new technique for minimizing distributed query time. In Proc. Of the Third workshop on Future Trends of Distributed Computing Systems, 1992

[CE86] T. Chao and C. J. Egyhazy, Estimating temporary file sizes in distributed relational database system, in Proc. Second Int. Conf and Data Engineering. pp. 4-12 1986

[CGS79] C. Baldissera, G. Bracchi, and S. Ceri. A query processing strategy for distributed databases, in proceedings of EURP-I.ip, pp 667-678 1979

[CHY97] M. Chen, H. Philip S. Yu On applying hash filters to improving the execution of multi-join queries, VLDB Journal vol 6 no.2 pp. 121-131 1997

[CL84a] A. L. P. Chen and V.O.K. Li, Deriving Optimal Semi-Join Programs for Distributed Query Processing. Proc. IEEE INFOCOM pp54-61 1984

[CL84b] A. L. P. Chen and V.O.K. Li. Optimizing Star Queries in a Distributed Database System. Proc. 10th International Conference On Very Large Data Bases, pp429-438 1984

[CKM80] C. T. Yu, K. Lam and M. Z. Ozsoyoglu. Distributed query optimization for tree queries. Dep. Inform. Eng. Univ. of Illinois at Chicago Circle, July 1980.

[CY93] M. Chen and P. S. Yu, Combining Join and Semijoin Operations for Distributed Query Processing, IEEE, Vol. 5. No. 3, 1993

[DPY84] D. W. Chiu, P. A. Bernstein, and Y. Ho, Optimizing chain queries in a distributed database system. SIAM J. Comput. Vol, 13, No 1 Feb 1984

[DY80] D. W. Chiu and Y. Ho. A methodology for interpreting tree queries into optimal semi-join expressions. In Proc. ACM SIGMOD Inc. Conference Management Data. pp169-178 1980

[EM78] R. Epstein and M. Stonebraker. Analysis of distributed database processing strategies, in Proceedings of the International Conference on Management of Data, Austin Tex. June 1978

[ES80] R. Epstein and M. Stonebraker, Analysis of distributed database processing strategies in proc. 6th Conf. On VLDB, Montreal, P.Q., Canada, pp. 92-101 1980

[GS82] N. Goodman and O. Shmueli, Tree queries: A simple class of relational queries, ACM Trans. Database Systems vol.7 no. 4, 177-187 1982

[G84] V. Gardarin. Join and semijoin algorithms for a multiprocessor database machine vol 9, no. 1 March 1984

[H79] A. R. Henver, The optimization of query processing on distributed database systems, Ph.D. dissertation, Database System Research. Center, Department Computer Science, Purdue Univ., W. Lafayette, IN, Rep. DB-80-02, Dec. 1979

[H79] A. R. Henver, The optimization of query processing on distributed database systems, Ph.D. dissertation, Database System Research Center, Department Computer Science, Purdue Univ., W. Lafayette, IN, Rep. DB-80-02, Dec. 1979

[IB86] A. Ijbema and H. Blanken, Estimating bucket accesses: A practical approach, in Proc. Second International Conference On Data Engineering. pp. 30-37 1986

[J82] Chang, J. A heuristic Approach to Distributed Query Processing, Proc. 8th Int. Conf. On Very Large Data Bases, pp 54-61 1982

[JM93] James K. Mullin, Estimating the size of a relational join, Information Systems Vol. 18. No. 3, pp. 189-196, 1993

[JM03] 60535 Class Notes by Joan Morrissey. 2003 Winter.

[KTY82] L. Kerschberg. P.D. Ting, and S. R. Yao. Query optimization in star computer networks. ACM Trans. Database Systems vol. 7 no. 4, 678-711 Dec.1982

[KR87] H. Kang, N. Roussopoulos, Using 2-Way Semijoins in Distributed Query Processing, IEEE, Number: TR 87-35 1987

[KR91] H. Kang, N. Roussopoulos, A Pipeline N-Way Join Algorithm Based on the 2-Way Semijoin Program, IEEE, Vol 3. No 4 pp 486-495 1991

[L83] G.M. Lohman, et al., On the design of a Distributed Query Processing Strategy, Proc. of the ACM SIGMOD Conf., pp. 30-39 May 1983

[LCC82] C. T. Yu, K. Lam, C. Chang and S. Chang. Promising Approach to Distributed Query Processing, Proc. 7th Berkeley Workshop on Distributed Data Management and Computer Networks, pp363-390 1982

[LT95] H. Lu and K. L. Tan. A framework for modeling cost in multidatabase. In Advances in Computing Techniques, Algorithms, Database and parallel Processing. JSPS – NUS Seminar on Computing, pages 114-123, 1995

[M81] M. Adiba. Derived Relations: A Unified Mechanism for Views, Snapshots and Distributed Data, Proc. 7th Int. Conf. On Very Large Data Bases, pp293-305 1981

[M90] J. K. Mullin. Optimal semijoins for distributed database systems. IEEE Transactions on Software Engineering, vol. 16, no. 5, pages 558-560, 1990

[ML86] L.F. Macket, G.M Lohman R* optimizer validation and performance evaluation for local queries, in Proc. 1986 ACM SIGMOD Conference, pp. 84-95 1986

[MOL00] J. Morrisey, WK. Osborn, and Y. Liang, Collisions and reduction filters in distributed query processing. 2000

[MP91] M.T. OzSu and P, Valduies, Principles of Distributed Database System, Prince Hall International, 1991

[OV99a] M.T. OZSU and P.Valdurize. Principles of distributed database systems. Second Edition, Upper Saddle River, NJ: Prentice-Hall, pp. 48 1999

[OV99a] M.T. OZSU and P.Valdurize. Principles of distributed database systems. Second Edition, Upper Saddle River, NJ: Prentice-Hall, pp. 254-259 1999

[OV99b] M.T. OZSU and P.Valdurize. Principles of distributed database systems. Second Edition, Upper Saddle River, NJ: Prentice-Hall, pp. 20-25 1999

[OV99c] M.T. OZSU and P.Valdurize. Principles of distributed database systems. Second Edition, Upper Saddle River, NJ: Prentice-Hall, pp. 239-247 1999

[P84] W. K. Perrizo. A method for processing distributed database queries. IEEE. Trans. Software Engineering. vol. 10 no. 4: 466-471 July 1984

[P85] W. K. Perrizo. Upper bound response time semijoin strategies. In 1st International Conference on Supercomputer Systems. St. Petersburg. Fla., pp. 273-279 Dec. 1985.

[PM79] P.G. Selinger and M. Adiba. Access Path Selection in a relational Database Management System: RJ2429, IBM Research Lab. San Jose, California, Jan 1979.

[PW81] P.A. Bernstein and D. W. Chiu. Using semi-joins to solve relational queries. J. ACM. Vol. 28. No. 1. Jan. 1981

[PW82] P. Black, and W. Luk, A New Heuristic for Generating Semi-Join Programs for Distributed Query Processing, Proc. IEEE COMPSAC pp116-123 1982.

[PWC+81] P.A. Bernstein, N. Goodman, E. Wong, C. Reeve, and J.B. Rothnie, Query Processing in a System for Distributed Databases (SDD-1), ACM Trans. Database System. Vol 6. Issue 4. pp 602-605 Dec. 1981

[RK91] N. Rossopoulous, H. Kang, A pipeline N-Way Join Algorithm Based on the 2-way Semijoin Program. Vol. 3, No. 4. pp. 486-495 1991

[S88] T.K. Sellis Multiple-Query Optimization, ACM TODS, VOL, 13, NO. 1, March 1988, 23-52

[SK+76] M. Stonebraker, P. Kreps, W. Wong, and G. Held. The Design and Implementation of INGRES. ACM Trans. Database Syst. Vol. 1 no. 3 pp.198-222 Sept 1976

[SL83] W. S. Luk, and L. Luk, Optimizing Semi-join Programs for Distributed Query Processing, Proc. 2nd Int. Conf. On Data Bases, 1983

[ST89] S. Salza and M. Terranova, Estimating bucket accesses: A practical approach, in Proc. ACM SIGMOD Conf., Porland, OR, pp. 8-14 1989

[SY80] G. M. Sacco and S. B. Yao. Query Optimization in Distributed Database Systems. Adv. Comput. Vol.21, Academic, New York, 1980

[TB95] T. Bealor Semi-join strategies for total cost minimization in distributed query processing, Windsor, Ont. University of Windsor, c1995

[TC92] J. C. R. Tseng and A. L. P. Chen. Improving distributed query processing by hash semijoins. Journal of Information Science and Engineering, vol. 8, pages 525-540, 1992

[VG84] P. Valduriez and G. Gardain. Join and semijoin algorithms for a multiprocessor database machine, ACM Trans. Database Systems vol. 9 no. 1 133-161 Mar. 1984

[W82] E. Wong, A Statistical approach to incomplete information in database systems, in ACM TODS, Vol. 7, no 3. pp. 470-488, Sept. 1982

[W85] W. Kim, Global Optimization of Relational Queries: A First Step, in Query Processing in Distributed Satabase Systems, Edited by Kim, Reiner and Batory, Springer – Verlag, pp. 207-216 1985

[WE77] E. Wong, Retrieving Dispersed Data from SDD1: A system for Distributed Databases, Proc. 2nd Berkeley Workshop on Distributed Data Management and Computer Network, 1977

[WY76] E. Wong and K. Youssefi. Decomposition: A Strategy for Query Processing. ACM Trans. Database System Vol. 1 no. 3, pp. 77-85 Sept. 1996

[Y85] C. T. Yu, Distributed database query processing, in Query Processing in Database Systems, W. Kim, D. Reiner, and D. Batory, eds. New York: Springer Verlag, pp. 48-61 1985

[Y86] C. Yu, et al., Adaptive Techniques for Distributed Query Optimization, Proc. of the Int. Conf. on Data Engineering, Los Angeles, CA, USA, pp. 86-93 Feb. 1986

[Y87] C. Yu et al, Algorithms to process distributed queries in fast local networks, IEEE Trans. Computer, vol. C-36, pp-1153-1163, Oct. 1987

[YC84] C. T. Yu and C. C. Chang, Distributed query processing, ACM Computing Surveys, vol. 16 no. 4 399-433 Dec. 1984

[YCB+86] C. Yu, C. Chang, D. Brill, and A. Chen, Adaptive techniques for distributed query optimization, in Proc. Fourth Int. Conf. On Data Engineering, Los Angeles, CA, FEB. pp. 86-93 1986

Vita Auctoris

Xiaobai Cao was born in Shanghai, China. She obtained Bachelor of Computer Information System in university of Windsor in 2001.Currently, she is a candidate for Master of Computer Science in University of Windsor, and will graduate in 2004 Summer.