2004

Implementation of composite semijoins using a variation of Bloom filters.

Yongmei Zhu
University of Windsor

Implementation of Composite Semijoins Using A Variation of Bloom Filters

By

Yongmei Zhu

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

Windsor, Ontario, Canada

2004

Your file Votre référence
ISBN: 0-612-92534-X
Our file Notre référence
ISBN: 0-612-92534-X

Abstract

Different from a centralized database system, distributed query processing involves data transmission among different sites and this communication cost is a dominant factor compared to local processing cost. So, the objective of distributed query optimization is to find strategies to minimize the amount of data transmitted over the network.

Since optimal query processing in distributed database systems has been shown to be an NP-hard problem, heuristics are applied to find a near-optimal processing strategy. Previous research has mainly focused on the use of joins, semijoins, and hash semijoins (Bloom filters). The semijoin is a commonly recognized operator, which provides efficient query results. As a variation of semijoin, the composite semijoin is beneficial to do semijoins as one composite rather than as multiple single column semijoins. The Hash semijoin (which uses a Bloom filter) is used to minimize the cost of a semijoin operation.

This thesis report provides a summary of each category of query processing techniques and optimization algorithms. Also in this thesis, we propose a new algorithm called Composite Semijoin Filter by combining the idea of composite semijoins, Bloom filters and PERF joins. One of the advantages of this algorithm is to avoid collisions. The algorithm is evaluated and compared with initial feasible solution (IFS) and another filter-based algorithm. It has been shown that the algorithm gives substantial reduction on relations and the total cost.

Key words: distributed query processing, semijoin, hash semijoin, composite semijoin, Bloom filter, PERF, query optimization, optimization algorithm

To my father, Songlin Zhu
my mother, Xueying Cao
my husband, Guanglei Li
my son, Shen Li

Acknowledgement

I am very happy to take this opportunity to convey my thankfulness to all people who helped me a lot during the whole process of my graduate study.

Firstly, I would like to thank my supervisor, Dr. Morrissey, who gave me instructive help and valuable advice. Without her patience and valuable guidance, it could be impossible to complete this thesis fluently. Besides, she is a so nice person who can be regarded as a sincere friend in study and life.

Secondly, I would like to thank my external reader Dr. Fung, my internal reader Dr. Lu and the committee Chair Dr. Ngom, for their time and instructive suggestions and comments.

Thirdly, I would like to thank all my friends and colleagues, who discussed with me on such topic and provided useful suggestions and experience during the research.

Finally, I would like to give my special thanks to my parents, my husband and my dear son, who gave me continuous support, encouragement and their endless love...

Table of Contents

List of Figures

Chapter 1 Introduction

Distributed database system technology is one of the major recent developments in the database system area. It is the outcome of the combination of database and computer network technology. So, in a distributed database system, the data is distributed and stored at different sites, which are connected by a computer network.

In distributed database systems, query processing plays an important role. An effective query (that means response time and total cost are all lowest) is a key factor affecting the system performance. Different from centralized query processing, distributed query processing involves data transmission among different sites and the communication cost is a dominant factor compared to local processing cost. As pointed out in [YC84], the process of a distributed query is composed of three phases:

1) Local Processing phase: All local processing operations such as selections and projections on the joining and target attributes are performed;
2) Reduction phase: Using optimization techniques and algorithms such as semijoins to reduce the size of relations in a cost-effective way, and thus reduce the total communication cost;
3) Final Query Processing phase: Send all resulting relations to the query site and reassemble them to generate the final query answer.

The main objective in distributed query optimization is to reduce the amount of data transmission. So, most research on optimization focuses on the reduction phase 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. Also, there are two cost models, which can be used to evaluate the performance of different algorithms. The response time version [SB82] considers that each operation is processed in parallel, so the response time is the maximum of the time from sending the query to getting the result is the cost of the processing. The total cost model [ESW78] considers the whole time consumed during processing. Since optimal query processing in distributed database systems has been shown to be NP-hard [BR88, PV88], heuristics are applied to find near-optimal strategies for query processing. Different relational operators (such as semijoin, 2-way semijoin, domain specific semijoin, composite semijoin, hash-semijoin and PERF join) and algorithms have been proposed. These approaches in distributed query processing have mainly been classified into the use of joins, semijoins, and hash-semijoins or combination of them. This report will give a summary of the research in this area.

The rest of this thesis is organized as follows: In Chapter 2, the literature review of distributed query processing and various strategies for distributed query optimization is described. Chapter 3 includes the motivation of my thesis and the proposed algorithm, Composite Semijoin Filter. The experiments and evaluation results will be given in chapter 4. Finally, we will give conclusions and work that will be done in the future.

Chapter 2 Literature Review

The contents of this chapter include an overview of distributed database systems, distributed query processing, query optimization techniques and algorithms.

2.1 Distributed Database System (DDBS)

In the field of data management, the developments in distributed computing technologies and network technologies lead up to distributed database management systems. A Distributed database (DDB) is a collection of multiple, logically interrelated databases distributed over a computer network [OZSU99, MTP99]. A distributed database management system (DDBMS) is a software system that permits the management of a distributed database and makes the distribution transparent to users. A distributed database (DDB) together with a distributed database management system (DDBMS) is called distributed database system (DDBS)[Vla97]. These systems should shield the users from the complexities of distribution. The advantages of a distributed database system are sharing data, stability and reliability because of replication. The main characteristics of a DDBS are:

- Transparent management of distributed and replicated data
- Reliability through distributed transactions
- Improved performance
- Easier system expansion

Users can gain many benefits from these advantages of a DDBS. But this will increase complexity and overhead and this additional complexity gives rise to new problems

influenced mainly by three factors [OV91][BG93], replication, tolerance and the synchronization of transactions. Query processing is one of the crucial problems in distributed database systems.

2.2 Distributed Query Processing (DQP)

An effective query will improve the system performance, especially in distributed database system environments.

2.2.1 What is DQP

Distributed query processing is a process to transform a high-level query language of distributed databases to a low-level database language for retrieving the database using an efficient and effective strategy. So, the problem is how to decide on a strategy for executing each query over the network in the most cost-effective way.

In a distributed environment, since data is geographically distributed, information has to be transmitted between sites in order to answer a query. So, in addition to the cost of centralized query processing, distributed systems face the problem of shipping data and results to and from sites. Usually, it is very expensive to move data from one site to another. For example, intermediate data derived at one site may need to be transferred to other sites for further processing, and the final result must be transferred to the query site. Cost may be acceptable on high performance local area networks, but not on others. So, the main factors to be considered are distribution of data and communication cost.

As mentioned above, a distributed database management system provides transparent access to distributed resources. There must be a module in the system architecture that gets a global query and manages a distributed evaluation. The whole distributed query process usually goes through three steps [OZSU99]: Parsing the global query, Query optimization and Query execution. Figure 2.1 shows the architecture of distributed query processing.

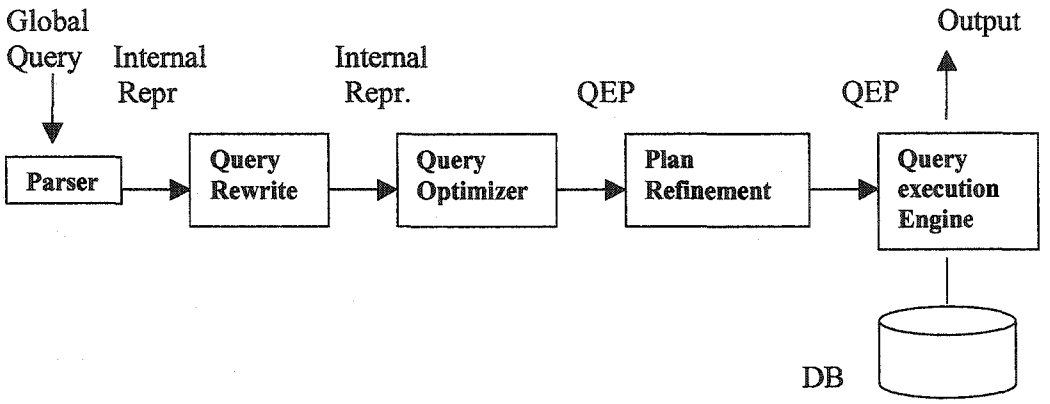


Figure 2.1 Architecture of Distributed Query Processing

When parsing the global query, each global query is replaced with a number of local queries according to the global schema. Then the query is simplified by eliminating redundant predicates. Finally, the query is transformed into relational algebra expressions. During the query optimization step, a distributed query execution plan (QEP) that obtains the answer is prepared. The execution plan says which local data are required, how to access them, which operations must be done at which sites. Moreover, the execution plan should be optimized, i.e., it minimizes the execution cost. Finally, a query execution engine in the query execution step executes the execution plan.

2.2.2 Cost Models for DQP

A cost model provides the basis for comparing different query execution plans (QEP) and for choosing the best plan for execution. The cost of distributed query processing can be expressed in terms of either the total cost model or the response time model. And costs are generally specified in terms of time units.

1) Total Cost Model

The total cost model considers the whole time consumed during processing. The total cost is the sum of all time incurred in the local processing and in intersite communication. In a distributed database system, the local processing costs include CPU and I/O cost, while communication cost is described in terms of the amount of data transmitted. A general formula [OZSU99] for total cost is:

$$\textbf{Total cost} = C_{CPU} * \#insts + C_{I/O} * \#I/Os + C_{MSG} * \#msgs + C_{TR} * \#bytes$$

- #insts is defined as the number of program instructions.
- #I/Os is defined as the number of transfers to or from disk.
- #msgs is defined as the number of messages transferred between one site and another.
- #bytes is defined as the total number of data sizes transmitted in all messages.
- C_{MSG} is the fixed cost of initiating and receiving a message.
- C_{TR} (transmission cost) is the cost of transmitting data between sites participating in the execution of the query.
- $C_{I/O}$ (Secondary storage Access cost) is the cost of loading data pages from secondary storage into main memory.
- C_{CPU} (Computation cost) is the cost of using the central processing unit (CPU).

The communication cost component is probably the most important factor considered in distributed database systems. However modern distributed processing environments have much faster communication networks whose bandwidth is comparable to that of disks. Therefore, more recent research efforts consider a weighted combination of all components since they all contribute significantly to the total cost of evaluating a query.

2) Response Time Model

The response time is the time from the initiation of the query to the time when the answer is produced. This model considers that each operation is processed in parallel as much as possible, so the maximum of the time from sending the query to getting of the result is the cost of the process. Since operations can be executed in parallel at different sites, the response time of a query may be significantly less than its total cost.

Minimizing response time can be achieved by increasing the degree of parallel execution. This does not imply that the total time is also minimized. On the contrary, it can increase the total time, for example, by having more parallel local processing and transmissions. So, in practice, a compromise between the two is sometimes desired.

2.2.3 Query Optimization Process

Query optimization is the process of ensuring that either the total cost or the response time of a query is minimized. Figure 2.2 shows the query optimization process [Fre89].

An input query is usually passed to the **Query Modification Module**, a stage that rewrites the initial query in order to improve efficiency during the evaluation of the

query. The input query can be represented by either relational algebra or a graph called a query graph. This query graph is then input to the **Query Execution Plan (QEP) Generator**, which defines how to create all possible QEPs from a query graph. Next, generated QEPs are submitted to the **Search Strategy Module** for deciding the best plan that gives the optimal cost from among the deferent QEPs. The **Cost Function** assigns a cost to each QEP selected by the search strategy module and provides the basis for comparing different QEPs and for choosing the best plan for execution.The optimal QEP is the plan that produces the cheapest cost. Query optimization is defined as the problem of finding the most efficient query execution plan (QEP) for a query expression.

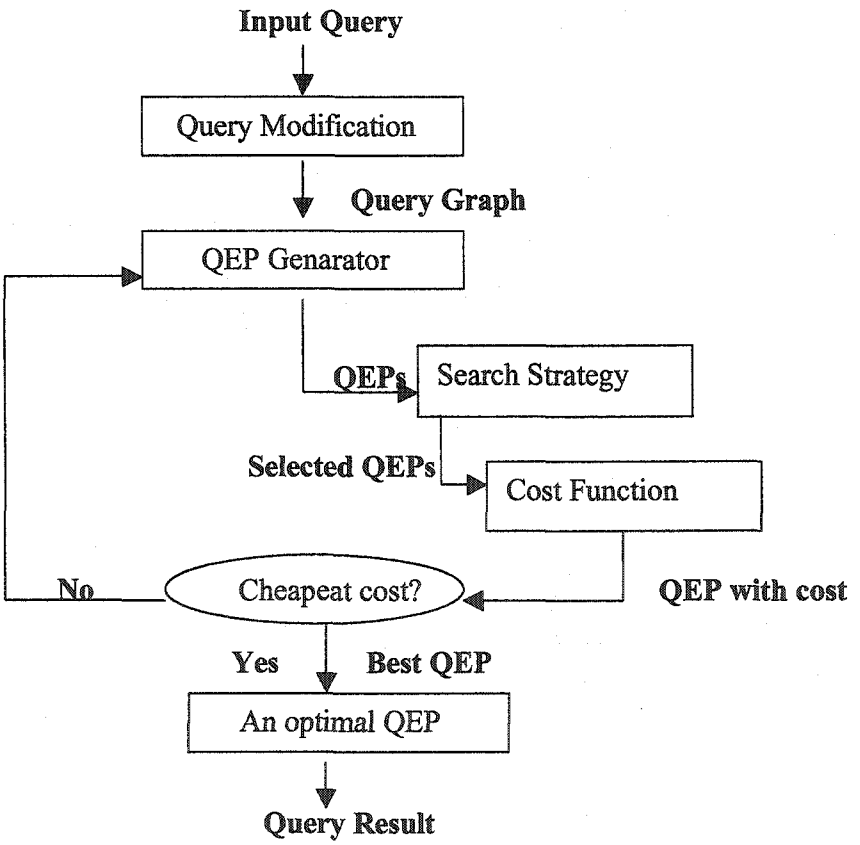


Figure 2.2 Query Optimization Process

Finding the optimal execution strategy for a query is NP-hard [BR88, PV88]. For complex queries with many relations, this can incur a prohibitive optimization cost. Therefore, the actual objective of the optimizer is to find a strategy close to optimal and, perhaps more important, to avoid bad strategies.

2.3 Query Optimization Techniques

One basic technique for reducing the amount of data transmission is the semijoin method. This approach increases local processing, but only a small projected portion of the relations is transferred during the reduction phase and only rows, which will participate in the final join, are transferred after the reduction phase. The Bloom filter method is similar to the semijoin. However, during the reduction phase, a bit vector carrying information about joining values is used. Also, some other techniques, such as 2-way semijoin [KR87], domain specific semijoin [CL90], composite semijoin [PC90] and PERF join [LR95] are introduced here.

2.3.1 Join

The join operator (?) is the most useful, the most commonly used and most simple way to reduce data. It can reduce the local processing cost and minimize the overhead of messages. In distributed database systems, because entire operand relations must be transferred between sites, the join operation becomes the main cost consuming process. So, the join operator is a time consuming operation.

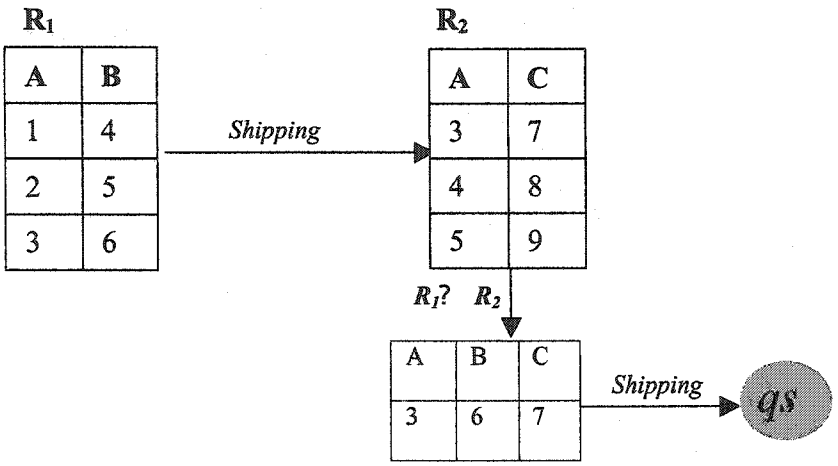


Figure 2.3 An example of join operation (R_1? R_2)

Figure 2.3 shows an example of join operation between relations R_1 and R_2 on the condition $R_1.A = R_2.A$. Whatever join operation is at site of R_1 or site of R_2, the whole relation should be shipped to the other site. And then the result may be transmitted to the query site. In this example the cost for transmitting relation R_1 to the site of R_2 is 6 (Suppose the cost of one data is one unit.).

2.3.2 **Semijoin** [BC81, BG81]

In distributed query processing, the semijoin is one of the most popular operators and has been used as an effective one, especially in reducing relations referenced in the query to reduce the total amount of data transmission between sites. It is obviously much less expensive to transmit the projected file than the entire file. Thus, it is often beneficial to reduce the size of relations through preliminary semijoins before transmitting the relation to the result site. Semijoin was first proposed by Berstein in [BC81] and [BG81].

A semijoin from R_i to R_j on attribute A can be denoted as $(R_i)_A$? R_j . It is computed in two steps:

1) Send projection of R_i on attribute A ($R_i[A]$) from site i to site j ;

2) Reduce R_j to R_j ' by eliminating tuples where attribute A is not matching any value in $R_i[A]$.

The cost of semijoin $(R_i)_A$? R_j is the size of projection of R_i on attribute A, while the benefit is difference of size R_j and R_j'. If the benefit exceeds the cost, then the semijoin is called a cost-effective.

Semijoin selectivity factor in R_i ? R_j is defined as the expected fraction of the tuples of R_i which belong to the result. Card (R_i ? R_j) =?*Card (R_j). An estimation of semijoin selectivity factor is: ? = Card ($R_i[A]$)/ Card (domain [A]).

Figure 2.4 shows an example of semijoin R_1 ? R_2 on attribute A.

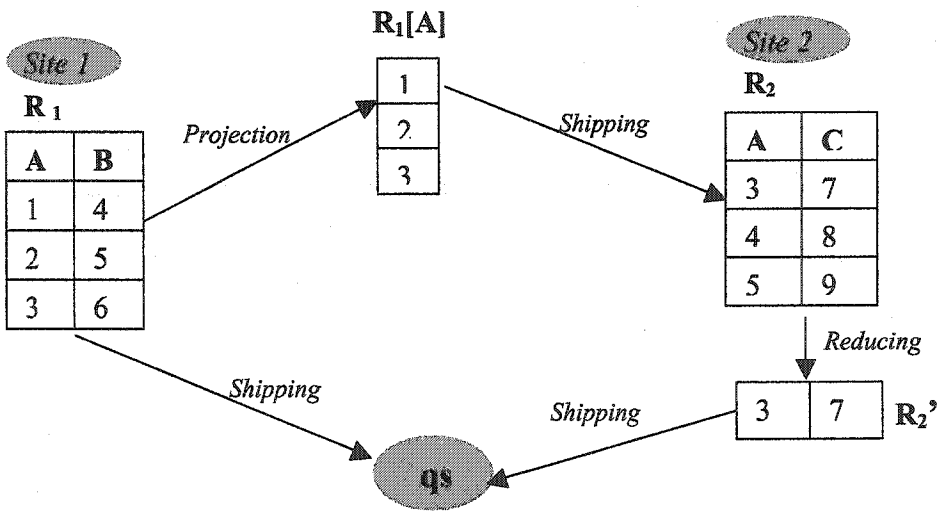


Figure 2.4 An example of semijoin (R_1 ? R_2)

First shipping the projection of R_1 (R_1 [A] ={1,2,3}) to site 2 then reducing R_2 to R_2 ' by eliminating tuples (crossed in the figure) where attribute A are not matching any value in R_1[A], only one tuple (3,7) left. In reassembling phase, R_1 and the reduced R_2 (R_2 ') may be transmitted to the query site (qs) and joined there to get the final result. The cost of this semijoin (C(s)) equals 3, benefit (B(s)) equals 4. Because the benefit exceeds the cost, this semijoin is cost effective.

2.3.3 2-way Semijoin [KR87]

As we talked about before, the semijoin acts as a size reducer for a relation much as a selection does and it is an effective operator to reduce the transmission of data. But it is a unary or a binary operator. That means it produces only one result relation. In [KR87], the author proposed a new relational algebra operator, called 2-way semijoin, which is an extended version of the semijoin. It has more reduction power than the semijoin and the propagation of the reduction effects is further than by the semijoin. These two aspects have been verified in [KR87].

A 2-way semijoin of R_i and R_j on attribute A can be denoted as $R_i \leftrightarrow R_j = \{R_i ?\ R_j,\ R_j ?\ R_i\}$. So, it can reduce R_i and R_j to R_i' and R_j' respectively. It is computed in the following steps [KR87]:

1) Send R_i [A] from site i to site j ;
2) Reduce R_j to R_j' by eliminating tuples whose attribute A is not matching any of R_i [A] and at the same time partition R_i [A] to R_i $[A]_m$ (match one of R_j [A]) and R_i $[A]_{nm} = R_i$ [A]- R_i $[A]_m$ (tuples in R_i not matching R_j [A]);
3) Send min (R_i $[A]_m$, R_i $[A]_{nm}$) back to site i ;
4) Reduce R_i to R_i ' using R_i $[A]_m$ (or R_i $[A]_{nm}$).

Figure 2.5 shows an example of 2-way semijoin ($R_1 \leftrightarrow R_2$). First ship the project of R_1 on attribute A ($R_1[A]=\{1,2,3\}$) to site 2; reduce R_2 to R_2' (tuples not matching $R_1[A]$ are eliminated), at same time $R_1[A]$ is partitioned into $R_1[A]_m = \{3\}$ and $R_1[A]_{nm} = \{1,2\}$; send $R_1[A]$m back to site 1; reduce R_1 to R_1 ' using $R_1[A]$m.

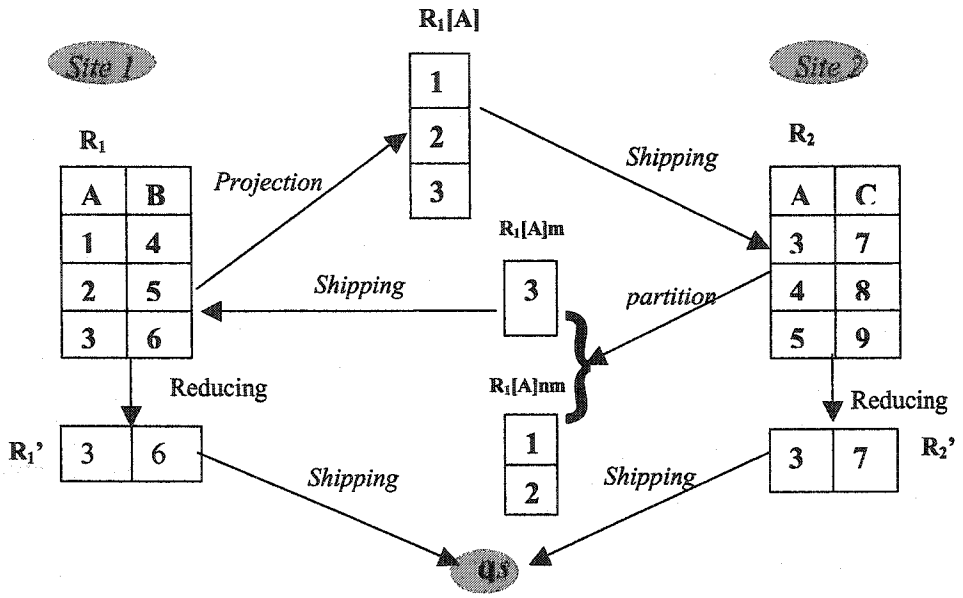


Figure 2.5 An example of 2-way semijoin ($R_1 \leftrightarrow R_2$)

The benefit of 2-way semijoin equals $[S(R_i) - S(R_i')] + [S(R_j) - S(R_j')]$ and the cost is $R_i[A]) + \min[S(R_i[A]_m), S(R_i[A]_{nm})]$. If the benefit exceeds the cost then it is called a cost-effective 2-way semijoin. In the above example, the benefit of 2-way semijoin $R_1 \leftrightarrow R_2$ equals 8; the cost of $R_1 \leftrightarrow R_2$ equals 4. So, this 2-way semijoin is cost effective.

The 2-way semijoin is always done in a cost effective way. Because no matter what the cost and benefit in the first two steps of computing a 2-way semijoin, the last two steps are always cost-effective. So, for two relations to be joined, even when only one or more can be reduced cost-effectively using semijoins, both may be reduced cost-effectively using a 2-way semijoin.

2.3.4 Pipeline N-way Join [RK91]

The semijoin is a mechanism that allows forward size reduction of relations and intermediate results. The 2-way semijoin enhances the semijoin with backwards size reduction capability for more cost-effective query processing. In [RK91], the author introduced a pipeline N-way join algorithm based on 2-way semijoins for joining the reduced relations residing on N sites. The main goal of this algorithm is to eliminate the need for shipping, storing, and retrieving foreign relations and/or intermediate results in the local disks of the query site during the processing of an N-way join. In the process, a structure known as *connector* (a small table, which can be easily fit in the memory for the next step semijoin) is in use, which records the former semijoin's effect.

The N-way pipeline algorithm proceeds in three phases [from RK91]:

1) Forward reduction & local processing phase:

   - Site of R_i receives from the site of R_{i-1} the projection of the joining attribute and constructs tuple connector C_i.

2) Backward reduction and collecting phase:

   - Backward reduction is applied to the tuple connectors not the relations.

   - A site containing R_i receives from the site of R_{i+1} the C_{i+1} tuple connector and joins it with its own C_i.

3) Pipeline executing phase:

   - The 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.

The advantages of the pipeline N-way join can be summarized as follows:

- No intermediate results are generated.
- Reduced relations are replaced by tuple connectors which are smaller in size; therefore, storing and transferring the tuple connectors is less expensive.
- The original relations are accessed once during the local processing phase.

2.3.5 **Interleaving Join with Semijoin** [CY90]

Although the join operator is a time consuming operation, judiciously applying join operations as reducers can further reduce the amount of data transmission required. Moreover, as pointed out in [CY90], the approach of combining join and semijoin operations as reducers can result in more beneficial semijoins due to the inclusion of joins as reducers (such semijoins are referred to as *gainful semijoins*.).

In [CY92], the author developed an efficient heuristic approach to determine an efficient sequence of semijoin and join reducers. First, obtain a sequence of join reducers and map it into a join sequence tree. In light of the join sequence tree, we derive important properties of beneficial semijoins. These properties are then applied to develop an efficient algorithm (G) to determine the beneficial semijoins that can be inserted into the join sequence. The experiments show that the approach of interleaving a join sequence with beneficial semijoins are not only efficient but also effective in reducing the total amount of data transmission required to process distributed queries [CY93].

2.3.6 Domain Specific Semijoin [CL90]

Many query optimization algorithms proposed for fragmented databases apply semijoins to reduce size of the fragments of joining relations before they are sent to a final processing site. When semijoins are employed in such a system, they have to be performed in a relation-to relation or a relation to fragment manner so that they will not cause the elimination of contributive tuples. So, semijoins cannot be performed between two fragments, because it may cause the elimination of some tuples before they are compared with all tuples of the other joining relation. In order to improve the semijoin operation associated with fragmented relations, the domain-specific semijoin is introduced in [CL90]. A domain specific semijoin can be defined as:

$$R_{ik} (A=B] R_{jm} = \{ r | r ? R_{ik} ; r.A ? R_{jm} [B] ? (Dom[R_i.B] - Dom[R_{jm}.B])\}$$

A and B are join attributes, R_{ik} and R_{jm} are two fragments of joining relation R_i and R_j respectively. A domain specific semijoin is computed in the following steps:

1) Calculate the estimated benefit and cost;
2) If it is found to be profitable, accept it in the current query-processing strategy; otherwise, ignore it;
3) Update the related information in the database profile.

Domain specific semijoin is based on many assumptions. It assumes all values of each attribute are randomly selected, all tuples are uniformly distributed over values of attributes and all values of attributes are independent. Experimental results [CL90] indicate that domain specific semijoins can reduce the size of fragments by eliminating

non-contributive tuples and can be performed in a fragment-to-fragment manner as in the application of regular semijoins and provide more flexibility in distributed query processing. It can also be shown that for a given query, there is always a strategy, using both domain-specific semijoins and semijoins, which is at least as good as the best strategy using only semijoin reductions.

2.3.7 **Composite Semijoin** [PC90]

In [PC90], Perrizo and Chen proposed a composite semijoin to minimize the response time for the queries. 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 with common source and common result sites. In this situation it may be beneficial to do the semijoins as one composite rather than as multiple single column semijoins. Through simulation results, it has been shown in [PC90] that algorithms including the possibility of composite semijoin can generate strategies, which are far better than those that ignore this method.

R_1

A1	A2	Nonjoin_ attrs
1	aa	~
1	bb	~
2	cc	~
3	cc	~

R_2

A1	A2	Nonjoin_attrs
1	cc	~
1	aa	~
2	bb	~
3	bb	~

Result Composite semijoinof R_1 and R_2

A1	A2	Nonjoin_attrs
1	aa	~

Figure 2.6 An example of composite semijoin R_1 and R_2

Figure 2.6 shows an example of a composite semijoin. Note that there is no reduction at all when single attribute semijoins are used, since all attribute A1 values are matched. However, there will be a significant reduction when a composite semijoin along attribute A1 and A2 is applied.

Also in [PC90], the author applied composite semijoins on some distributed query processing algorithms, such as algorithm GENERAL [AHY83] which is used to minimize the response time for general queries and produces total time strategies which are quite efficient and algorithm W [PC90] which guarantees a least bound response time for queries. Experimental results indicate that including the possibility of composite semijoins in a query-processing algorithm is likely to result in substantial response time reduction. It can be verified that the strategy formed by the algorithms, which apply composite semijoin, is always as good as the strategy of not allowing the algorithm to optimize for composite semijoin.

2.3.8 Hash Semijoin (Bloom Filter) [TC92]

In [TC92], the author proposed a new relational operator, called Hash semijoin, to minimize the cost of a semijoin operation (i.e., the cost of transmitting the semijoin projection). Hash semijoin is designed based on the concept of a search filter (also called a Bloom filter). Bloom filters are used to filter out the tuples that do not participate in the join. A bloom filter is a vector of bits, which represents the semijoin projection.

The hash semijoin of R_i and R_j is denoted R_j? R_i. It is computed as follows:

Step 1: Initialize a bit array of F bits to all is 0. The size of F is computed by

$F=(d/\ln 2)*|R_i|$;

Step 2: For each value of the join attribute in R_i, generate *d* bit addresses using the *d* hash functions and set the corresponding *d* bits in the bit array to 1 (i.e., set *F[k]= 1* if there exists join attribute value *v* in relation R_i, such that $d(v)=k$) ;

Step 3: Transmit the bit array to the site of R_j;

Step 4: For each tuple of R_j, use the *d* hash functions to hash the join attribute value to *d* bit addresses. Test if all the *d* bits in the bit array are 1s. If Yes, output the tuple to the result relation R_j', else discard the tuple.

Figure 2.7 shows an example of hash-semi-join (R_2? R_1) operation with the perfect hash function (H (x)=x).

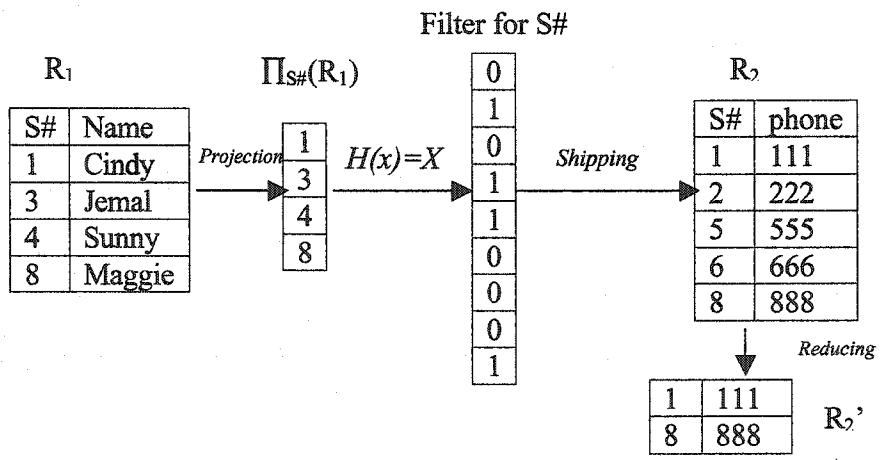


Figure 2.7 An example of hash semijoin (best HSJ)

It is obviously that hash semijoins have less cost of transmitting the filter than that of transmitting the semijoin projection in traditional semijoin. But *false drops* may occur,

which the search filters falsely accepts a value. *False drop probability* is used to refer to the probability that a false drop occurs. It depends on the size of the bit array (*F*), and the number of hash functions *d*. Figure 2.8 shows an example of hash semi-join with false drop. In this example, the hash function is changed to H (x) = x mod 5.

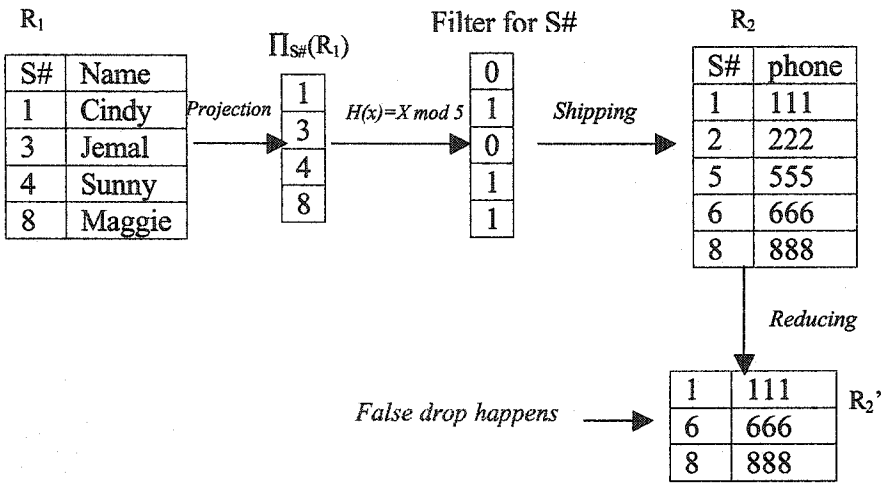


Figure 2.8 An example of hash semijoin (with false drop)

In this example, the hash function is changed to H(x) = x mod 5. After using the hash function to hash the value of attribute S# in R_1, the filter will be F={0,1,0,1,1}. When the filter is shipped to the site of R_2 and the same hash function is applied to the value of S# in R_2: H(1) = H(6) = 1, H(8) = 3, the result relation has three tuples. But the tuple (6,666) is falsely accepted by the search filter. One of the solutions is by increasing the number of hash function, but the cost may also increase. So, in practice, the number of hash functions is a key factor in hash semijoin operation.

Compare to semijoin, the cost, benefit and potential cost are given as following table.

	Semijoin	**Hash-semijoin**
Cost	$CT_{ij}=a_{ij}\|Ri\|$	$CH_{ij}=(d/\ln 2)*\|Ri\|$
Benefit	$BT_{ij}=(1-s_{ij})w_j\|R_j\|-a_{ij}\|Ri\|$	$BH_{ij}=(1-s_{ij}-f)w_j\|R_j\|-(d/\ln 2)*\|R_i\|$
Pcost	$PCT_{ij}=a_{ij}\|R_i\|+s_{ij}w_j\|R_j\|$	$CH_{ij}=(d/\ln 2)*\|R_i\|+s_{ij}w_j\|R_j\|$

Figure 9 Comparison between semijoin and hash-semijoin

- $|R_i|, |R_j|$ *-- cardinality of* R_i, R_j
- a_{ij} *– width of the join attribute*
- w_j *-- The width of a tuple in* R_j
- s_{ij} *– the selectivity of semijoin* R_j ? R_i ; f *– false drop probability*

It can be proved in [TC92] that hash-semijoin is more cost-effective than semijoin; the search filter in the hash-semijoin achieves considerable savings in the cost of a semijoin operation and the replacement algorithm can produce a more cost-effective semijoin program.

2.3.9 PERF Join [LR95]

In [LR95], Li and Ross present "Positionally Encoded Record Filters" (PERFs) and describe their use in a distributed query processing technique called PERF join. A PERF is a novel two-way join reduction implementation operator. This method adds to semijoins in the backward phase and is used to eliminate unnecessary redundant semijoins by using bit vectors. It is based on the relation tuple scan order instead of hashing. Hence, it does not suffer any loss of join information incurred by hash collisions. The basic idea of the PERF join is as follows: as in 2-way semijoin R? S, relation S is reduced by a semijoin with the projection of relation R (P_R). But instead of

transmitting P_s' back to R, send a bit vector (PERF) that contains one bit for every tuple in P_R. That bit is set to 1 if it is in P_S' and 0 otherwise. The order of the bits in the bit vector is the same tuple order of P_R that R's site sent initially. Consider two relations R and S, the steps of PERF is as follows:

1. Project R on A joining attribute (P_R);
2. Ship P_R to S;
3. Reduce S by a semi-join with P_R;
4. Send back to R a bit vector (PERF) that contains one bit for every tuple in P_R and in the same order. If the tuple is matching then send 1 else send 0.

The main utility of PERF is that it minimizes this phase and hence makes the forward phase (step2) cost greater than the backward phase. PERF joins can be better enhanced by sending back to R not all the bit vector corresponding to P_R but only the 0s part or 1s part according to which one is less in size and hence has lower transmission cost. As an example, figure 2.9 shows two PERFs for relation R (A, B) and S (A, C).

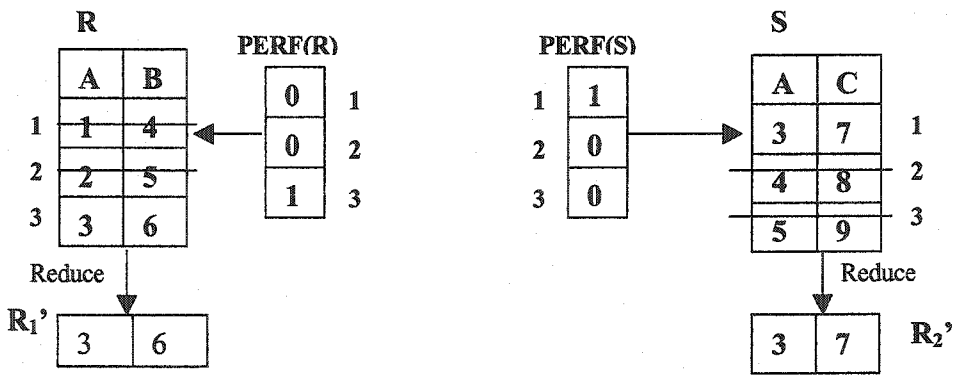


Figure 2.9 An example of PERF join

In the example, first, ship P_R on attribute A to site of S and reduce S with P_R, then send back to R a bit vector PERF(R) (one value (3) is matching, so the bit vector is {0,0,1}), finally reduce R to R', the same for S.

PERF join is a competitive alternative to 2-way semi-join and Bloom join. Analytical studies show that the response time of distributed join query processing algorithms can be improved by employing PERFs and PERF joins instead of or in addition to the traditional Bloom join and semi-join variant. The features of PERF-based techniques are:

- Preservation of complete join information.
- Minimal network and storage overhead.
- Cheap local join processing cost, especially when buffer memory scarce.
- Inequalities join query handling.
- Cyclic join query handing.

2.3.10 **Virtual Join** [SSL+02]

Virtual join [SSL+02] considers reducing both communication cost and local cost in distributed query processing. The basic idea of virtual join is to execute a join query through "discussion". During the discussion, the participating sites use tiny pieces of data to exchange their information. So it is much smaller than the real result and it can reflect the cardinality of the real result. It makes each site obtain the knowledge of the final result, and it filters out useless tuples at each site. The physical format of the knowledge is called virtual result in the sense that it can represent the final result. From the virtual result, the materialized result can be built easily.

For example, in figure 2.10 relation R and S are joined by attribute X. For join R? S on attribute X, V_{RS} is formally defined as a table with three fields. The first field is the join attribute, and the other two fields contain the number of useful tuples from R and S. For each value *y* of the first field, the other two fields contain the number of tuples that has *X* value = *y* from R and S respectively. V_{RS} describes the structure of the real result. It remains the information for both further joins and final assembly. For virtual result, the cardinality of tuples in the real result can be easily calculated. (4*2+1*3=11)

R

X	R other
B	...
A	123
D	456
A	...

S

X	S other
A	xxx
K	...
3	mmm
A	...

V_{RS}

X	R	S
A	4	2
D	1	3

Figure 2.10 Virtual result in joining R? S

It has two desirable features: 1) Being adaptive to different values of selectivity. 2) Giving accurate cardinality of the join result before it is materialized. Experiment results showed virtual join was both adaptive and efficient [SSL+02].

2.4 Query Optimization Algorithms

The objective of distributed query optimization is to find a query optimization algorithm to generate an optimal processing strategy in the solution space of all possible execution strategies. There are three types of optimization [Kos98]: Exhaustive search approach, Randomized strategies and Heuristics. The efficiency of processing strategies for queries in a distributed database is critical for system performance. If a query is processed inefficiently, it not only takes a long time before the end user gets his answer, but it might also decrease the performance of the whole system. Many methods have been studied to minimize the response time or the total cost. They can be classified into join-based, semijoin-based and filter-based or combination-based. It has been proven that finding the optimal solution is NP-hard [BR88, PV88]. So, generally, we only try to develop algorithms, which are efficient but perhaps only near optimal. In this section, some main algorithms are introduced.

2.4.1 Join-Based Algorithms

Join ordering is an important aspect of centralized query optimization. In a distributed database system, it is even more important since joins between relations may increase the communication cost over a network. Some algorithms optimize the ordering of joins directly without using semijoins. Distributed INGRES [ES80] and System R* [SA80] algorithms are representative of algorithms that are based on joins. INGRES employs a dynamic optimization algorithm, while System R* uses a static optimization.

1. Distributed INGRES [ES80]**:** This algorithm is based on a heuristic approach. The objective of the algorithm is to minimize a combination of both the communication time and the response time. However, these two objectives may be conflicting. For instance, increasing communication time may well decrease response time. Thus, the function can give a greater weight to one or to the other. The algorithm is called D-INGRES-QOA.

The input of this algorithm is a multi-relation query (MRQ) expressed in tuple relational calculus, the output is result of the last multi-relation query (MRQ'). It is computed in three steps:

Step1: All mono-relation queries (e.g., selection and projection) that can be detached are first processed locally;

Step2: Execute the reduction algorithm [ES80] to produce a sequence of irreducible subqueries, with at most one relation in common between two consecutive subqueries;

Step3: Chooses next irreducible subquery involving the smallest fragments; selects the best strategy to process the query; and transfers all fragments to their processing sites; finally executes the query. Repeat this step until there are no remaining subqueries left.

For example, assume that relations EMP (is fragmented into EMP_1 and EMP_2), ASG and PROJ of the query are stored as follows: EMP_1 and ASG are stored at site 1; EMP_2 and PROJ are stored at site 2. There are several possible strategies, including the following:

1) Execute the entire query (EMP ? ASG ? PROJ) by moving EMP_1 and ASG to site 2;

2) Execute the entire query (EMP ? ASG) ? PROJ by moving (EMP_1 ? ASG) and ASG to site 2, and so on.

The choice between the possible strategies requires an estimate of the size of the intermediate results. For example, if *size* (EMP_1 ? ASG) > *size* (EMP_1), strategy 1 is preferred to strategy 2. Therefore, an estimate of the size of joins is required.

The algorithm of distributed INGRES is characterized by a limited search of the solution space, where an optimization decision is taken for each step without concerning itself with the consequences of that decision on global optimization. However, the algorithm is able to correct a local decision that proves to be incorrect. An alternative to the limited search is the exhaustive search approach, where all possible strategies are elevated to find the best one. In [ES80], the two approaches are simulated and compared on the basis of the size of the data transfers. The study shows that exhaustive search significantly outperforms limited space as soon as the query accesses more than three relations and dynamic optimization is beneficial because the exact sizes of the intermediate results are known.

2. System R* [SA80] performs static query optimization based on an exhaustive search of all alternative strategies of the solution space, in order to choose the one with the least cost. The optimizer of the master site makes all intersite decisions, such as the selection of the execution sites and the fragments as well as the method for transferring data, while the apprentice site makes the remaining local decisions and generates 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 communications cost.

The input to the algorithm is a localized query expressed as a relational algebra tree (QT), the location of relations and their statistics. After executing the procedure R*- QOA, a minimum cost strategy will be generated. R*- QOA is executed in three steps:

1) For each relation R_i in the query tree QT, find its best access path which has minimum cost;
2) For each order, build strategy (semijoin sequence) with minimum cost;
3) For each site k storing a relation involved in QT, generate its local strategy (LS_k).

To join two relations, there are three candidate sites: the site of the first relation, the site of second relation, or a third site. In R*, two methods are supported for intersite data transfers.

1) Ship-whole. The entire relation is shipped to the join site and stored in a temporary relation before being joined.
2) Fetch-as needed. The external relation is sequentially scanned, and for each tuple the join value is sent to the site of the inter-loop relation, which selects the internal tuples matching the value and sends the selected tuples to the site of the outer-loop relation.

The trade-off between these two methods is obvious. Ship-whole generates a larger data transfer but fewer messages than fetch-as-needed. It is intuitively better to ship whole relations when they are small. On the contrary, if the relation is large and the join has

good selectivity (only a few matching tuples), the relevant tuples should be fetched as needed. R* does not consider all possible combinations of join methods with transfer methods since some of them are not worthwhile.

2.4.2 Semijoin-Based Algorithm

A semijoin program is a sequence of semijoins generated by the query optimizer. The objective of query optimization is to find an optimal semijoin program, which requires the least total transmission cost to process the query. To generate the efficient semijoin program, numerous algorithms have been devised, especially for some special classes of queries such as simple queries [AHY83], chain queries [CBH84], star queries [CL85] and tree queries [PV88, Won90]. Most existing algorithms are heuristics. In this section, we will introduce some representative algorithms based on semijoin.

1. SDD-1 algorithm [BGW+81] is the first method in distributed query processing using semijoin as reducer to minimize the cost. It is based on hill-climbing strategy [Won77] by replacing join with semijoin. The main step of the algorithm consists of determining and ordering beneficial semijoins whose costs is less than their benefits. It proceeds in four phases: initialization, selection of beneficial semijoins, assembly site selection, and post optimization.

Initialization phase generates a set of beneficial semijoins (BS) and execution strategy (ES) that includes only local processing; the second phase selects the beneficial semijoins from BS by iteratively choosing the most beneficial semijoin and modifying the database statistics and BS accordingly. The iterative phase terminates when all semijoins in BS

have been appended to the execution strategy. The order in which semijoins are appended to ES will be the execution order of semijoin; The third phase selects the assembly site by evaluating, for each candidate site, the cost of transferring to it all the required data and taking the one with least cost; Finally a post-optimization phase permits the removal from the execution strategy of those semijoins that affect only relations stored at the assembly site. A general outline of the SDD-1 algorithm (OPT) is given as follows [BGW+81]:

1) Maps a query into an envelope. An envelop is a relational calculus expression that maps a database into a sub-database;
2) Evaluates the envelop and translates it into reducer. A program contains relational operations and performs the reduction of the relation size;
3) Execute the query at a site using the data assembled by 2) step.

OPT is a greedy optimization algorithm, it always seeks to maximize immediate gain. It never looks ahead, and never backs up. In general, it is sub-optimal. SDD-1 optimization algorithm is designed under the assumption that relations can be transmitted to another site. This is not true for those relations that have been selected after beneficial semijoins are considered. The algorithm only selects semijoins that maximize immediate gain, not considering the fact that execution of one semijoin might affect the performance of the other semijoins. Therefore, the drawback is that some semijoins may incorrectly be considered beneficial in SDD-1.

2. AHY: In [AHY83], Apers, Hever and Yao introduced and investigated a family of optimization algorithms using semijoins to minimize either the response time (algorithm PARALLEL) or the total time (algorithm SERIAL) and extended those algorithms to

algorithm GENERAL that processes general distributed queries. The main idea of the algorithms is to reduce the sizes of each relation by possible restrictions and projections instead of computing the joins immediately. If one relation has the join attributes, we use semijoin to delete the unnecessary tuples. For example: Attributed d_{21} (d_{ij} represents attribute j of relation R_i.) is sent to attributed d_{31}, a semijoin is performed on relation R_3. The reduced d_{31} can be sent to relation R_1 in parallel. Finally the reduced relation R_1 is sent to the result node. Here is a summarization of these algorithms [AHY83].

There are four steps in algorithms **GENERAL** [From AHY83]:

1) Finish 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 result node. Algorithm PARALLEL is called to minimize response time; Algorithm SERIAL is called to minimize total time. This results in one schedule per simple query.
3) Integrate candidate schedules. For each relation, candidate schedules are integrated to form a processing schedule. The integration is done by procedure RESPONSE for response time minimization and by procedure TOTAL or COLLECTIVE for total time minimization.
4) Remove schedule redundancies, whose relations have been transmitted.

Algorithm **PARALLEL** is used to minimize response time by searching for cost beneficial data transmissions in the current system state s_i, selectivity $?_i$ and schedule response time r_i of each relation R_i. The selectivity $?_i$ of an attribute is defined as the number of different values occurring in the attribute, divided by the number of all

possible values of the attribute. The algorithm can be described as follows [From AHY83]:

1) Order R_i, (s_1=...= s_m) in ascending order of size;
2) For each R_j (j <i) construct a schedule to R_i that consists of parallel transmission of R_j and all schedules of R_k (k<j). Select schedule with minimum response time.

Algorithm **SERIAL** is used to minimize the total time. It is executed in three steps.

1) Order relation R_i such that $s_1 = s_2 = \ldots = s_m$;
2) If no relations are at the result node, then select strategy: R_1? R_2 ? ... ? R_n ? ... ? R_r or else if R_r is a relation at the result node, then there are two strategies:

R_1? R_2 ? ... ? R_r? ... ? R_n ? ... ? R_r or R_1? R_2 ? ... ? R_{n-1}? ... ? R_n ? ... ? R_r.

3) Select the one with minimum total time.

Procedure **RESPONSE**

1) Candidate schedule ordering in ascending order of arrival time;
2) Schedule integration: for each candidate schedule, construct an integrated schedule for the relation that consists of the parallel transmission. Then select the integrated schedules with minimum response time.

Procedure **TOTAL**

1) Adding candidate schedule;
2) Select the best candidate schedule;

3) Candidate schedule ordering;

4) Schedule integration.

Procedure **COLLECTIVE**

1) Select candidate schedule with minimum cost and selectivity less than 1;

2) Build processing strategy for parallel transmission;

3) Test variation of strategy.

3. Algorithm W [MB96] is a static strategy with two distinct phases: first, a schedule of semijoins is established using a cost/benefit analysis which is based on estimates of the selectivities of the attributes and the sizes of intermediate results; second, the schedule is executed.

Phase one: Establish the schedule. [From MB96]

Step1: Consider how a reducer might be built for each join attribute.

1) Sort the attributes so that $|d_{aj}| = |d_{bj}| = \ldots = |d_{mj}|$;

2) Evaluate the semijoins in order beginning with d_{aj} ? d_{bj}. this semijoin is appended to the schedule if it is gainful. If the semijoin is appended then d_{bj}* ? d_{cj} is evaluated next, otherwise d_{aj} ? d_{cj} is evaluated. This process is repeated until all the semijoins in the sequence have been evaluated.

Step2: Examine how each reducer might be used. In this step we examine how the relation sizes would be changed by the construction and use of the reducers, in the order from smallest to largest.

1) Sort the reducers, from smallest to largest;
2) For each reducer in turn, estimate the reduction effects of constructing and using it. Profitable semijoins are appended to the schedule.

Step 3: Look for remaining profitable semijoins.

1) Sort the attributes by increasing size;
2) Evaluate each semijoin in the sequence, appending it if it is profitable.

Phase two: Construct the reducers and ship them to the designated sites for semijoining and finally the reduced relations are transferred to the query site where the answer is assembled.

Compared to AHY algorithm, Algorithm W works well as a method of reducing the total amount of data transferred over the network during processing. Experimental results show that in all cases Algorithm W is superior but on average Algorithm W outperforms AHY by 17% [MB96]. And there are no synchronization problems or difficulties with redundant transmissions in Algorithm W, while there are in the AHY Algorithm.

4. Improvement algorithms for semijoin

In [CL84], the authors identified four properties that optimal semijoin programs for processing tree queries have to satisfy. A semijoin program is represented by an

execution graph, which specifies the order and the identities of the semijoins to be executed. Given a semijoin program, we can therefore apply these properties to check its optimality. If it does not satisfy these optimality properties, the associated improvement algorithms can be applied to improve this program.

Property 1: No redundant semijoin occurrences. If there exist redundant semijoin occurrences then delete all redundant semijoin occurrences and resultant isolated nodes;

Property 2: The execution graph of an optimal semijoin program cannot be rearranged by the rearrangement techniques.

Property 3: Each NSJ(necessary semijoin) is properly embedded in the optimal semijoin program.

Property 4: Each end node of the execution graph of an optimal semijoin program must be a final relation.

Four algorithms which apply the optimality properties are presented to check the optimality of a give semijoin p and improve it when possible.

Algorithm P1 is based on optimality property 1 and is used to delete redundant semijoin occurrences and resultant isolated nodes; Algorithm P2 is based on property 2 and applies rearrangement techniques if p can be rearranged; Algorithm P3 is based on property 3; Algorithm P4 is based on property 4 and is used to delete non-final relations, if each NSJ with Y (the tree rooted at a final relation) is properly embedded in p then repeat to delete the semijoin occurrence whose successor node is an end node in the execution graph of

the semijoin program and is a non-final relation until every end node in the execution graph of the semijoin program is a final relation.

2.4.3 Combination-Based algorithm

In distributed query processing, the conventional approach to reduce the amount of data transmission is to apply a sequence of semijoins as reducers and then ship the reduced relations to the final site to execute the join operation. As pointed out in [CY90], judiciously applying join operations as reducers can lead to further reduction in data transmission. The combination-based algorithm is executed in two phases.

In the first phase, an algorithm G [CY90] is used to determine beneficial semijoins for a join sequence. If we use SM_T to represent the set of possible semijoins and SM_J to represent the beneficial semijoin, the algorithm G can be summarized simply as follows: [From CY90]

- Determine SM_T from the query graph;
- Sort the semijoins in SM_T in a descending order of their cumulative benefits;
- Set initial of SM_J is empty;
- If a semijoin in SM_T is beneficial, then insert the semijoin to SM_J.

In phase two, the identified beneficial semijoins can be inserted into the join sequence according to the procedure P. The following are the general steps for procedure P:

Step 1: In the join sequence tree, perform join operations associated with leaf nodes that are neither reducers nor reducers of the semijoins in SM_J. Update the join sequence tree by merging the leaf node to its parent node after each join operation is performed.

Repeat Step 1 until there is no such join available.

Step 2: If there is a semijoin SJ_i in SM_J, the reducer is a leaf node of the join sequence tree, then perform SJ_i, remove SJ_i from SM_J, and go to Step 1, otherwise, go to Step 3.

Step 3: Choose a semijoin SJ_k with the smallest cost from SM_J. Perform SJ_k and remove it from SM_J. Go to Step 1.

In [LC01], semijoins and joins are termed contributive replicated semijoins and contributive replicated joins, respectively, when they are interleaved into a join sequence to reduce the amount of data transmission cost required in a network with replicated relations. The solution procedure consists of three consecutive steps, namely relation selection, join sequence scheduling and merge processing. A simulator is developed to evaluate the performance of algorithms devised. The results show that the approach of interleaving a join sequence with contributive replicated semijoins/joins is not only efficient in its execution but also effective in reducing the total amount of data transmission cost required to process distributed queries.

2.4.4 Filter-Based Algorithm

A filter-based algorithm named the Replacement Algorithm is proposed in [TC92]. The input of the algorithm includes the number of hash functions d and the double linked list T representing the execution tree. The algorithm will output an improved semijoin program. The general steps for a backward replacement with hash-semijoin can be described as follows: [From TC92]

1) Establish a queue that is used to record the nodes having no successors with their levels and then sort the queue by levels of nodes from high to low;
2) Remove the first element (denoted $R_{j)}$ from the queue, and its predecessor is R_i;
3) Calculate the potential cost of traditional semijoin (R_i ? R_j) and hash semijoin (R_i? R_j);
4) If hash-semijoin is more cost effective than semijoin, then replace the semijoin R_i ? R_j by hash semijoin R_i? R_j
5) Update the potential cost of R_i;
6) Insert the element (R_i, level) into the queue according to the level of R_i;
7) Repeat the process until the queue is empty.

Here is an example to show how this algorithm works. The semijoin program

R_2 ? R_1, R_3 ? R_2, R_4 ? R_2, R_5 ? R_4 is represented by the execution graph shown in figure 2.11.

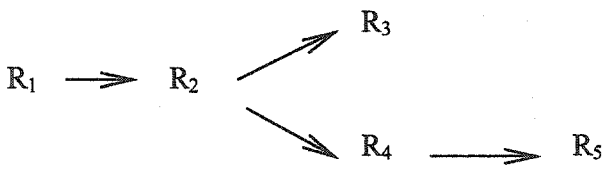


Figure 2.11 An example of execution graph

Suppose: $|R_i| = 1000$, $w_{ij} = 100$, $s_{ij} = 0.5$, $a_{ij} = 30$ bits, $d = 5$,then

- *false drop probability (f)* $= (1/2)^d = (1/2)^5 = 0.03125$
- potential cost of semijoin (CT_{ij})$= a_{ij}|R_i| + s_{ij}w_j|R_j| = 80000$
- potential cost of hash-semijoin (CH_{ij})$=(d/\ln 2)*|R_i| + (s_{ij}+f)w_j|R_j| = 60338$

Initialization: establish and short the queue (SQ) = {(R_5, 4), (R_3, 3)}; the potential cost C_i of the subtree rooted at R_i is 0 (i=1 to 5);

Execution: •Remove (R_5,4) from SQ, R_j=R_5, R_i=R_4

•Calculate CT_{45}-CH_{45}-f •C_5

= 80000 – 60338 – 0.03125 •0

= 19662 >0, so replace R_5 ? R_4 by R_5 ? R_4 and

Upda te C_4=C_4 + CH_{45} +(s_{45}+f) •C_5 = 60338

•Insert (R_4, 3) to SQ, SQ = {(R_3,3),(R_4,3)}

Repeating the execution until SQ is empty. In this example, all semijoins are replaced by hash-semijoin.

The potential cost of the original semijoin program:

$$C_0 = CT_{12} + s_{12} \bullet CT_{12} + s_{12} \bullet CT_{24} + s_{12} \bullet s_{24} \bullet CT_{45}$$

=80000+0.5 •80000+0.5 •80000+0.5 •0.5 •0.5 •80000

= 180000

The potential cost of the improved semijoin program is: $C_1 = 141476$

So, in this example, the cost is saved $C_0 - C_1 = 38524$.

In general, hash semijoin is more cost-effective than semijoin. The search filter in the hash-semijoin achieves considerable saving in the cost of a semijoin operation. However, it only works on execution tree, and the performance is tightly related with the hash functions.

2.5 Conclusions

Distributed query processing is the process of retrieving data from different sites. It involves transmission via a network so it will create delays. The basic challenge is to design and develop efficient query processing techniques and strategies to minimize the communication cost.

- The semijoin [BC81, BG81] is a very popular technique for reducing transmission cost. Most distributed query processing algorithms proposed so far rely on semijoin;
- The 2-way semijoin is an extension of the semijoin operation [KR87]. It aims to reduce both relations, while requiring less total network cost than executing regular semijoin;
- The pipeline N-way join is for joining the reduced relations residing on N sites. The main goal is to eliminate the need of shipping, storing, and retrieving foreign relations and/or intermediate results in the local disks of the query site [RK91];
- Interleaving joins with semijoins can result in more beneficial semijoins due to the inclusion of joins as reducers. Judiciously applying the join operator as reducer can further reduce the amount of data transmission required [CY92];
- The Domain-specific semijoin can reduce the size of fragments by eliminating non-contributive tuples and can be performed in a fragment-to-fragment manner and provide more flexibility in distributed query processing [CL90];
- A composite semijoin is a semijoin in which the projection and transmission involve multiple columns. It may be beneficial to do the semijoins as one composite rather than as multiple single column semijoins [PC90];

- Hash semijoin transmits a Bloom filter that is a hashing based bit vector used to encode the same joining information as the join attribute projections do. Compared to semijoin, hash semijoin can have lower cost because a bloom filter is generally smaller than the join attribute projection, but false drops may occur [TC92];
- A PERF method adds to semijoins in the backward phase and is used to eliminate unnecessary redundant semijoins by using bit vectors. It is based on the relation tuple scan order instead of hashing. Hence it does not suffer any loss of join information incurred by hash collisions [LR95].
- Virtual join [SSL+02] considers reducing both communication cost and local cost in distributed query processing. It is both adaptive and efficient.

It has been shown that finding an optimal query strategy for a given query is NP-hard, so most research concentrates on developing heuristic algorithms which find near-optimal solutions. [Kos98] presents that all query optimization algorithms fall into one of three different classes (Exhaustive search, Heuristics, randomized algorithms) or combinations of such basic algorithms.

- Distributed INGRES [ES80] and System R* [SA80] algorithms are two representative of algorithms that are based on joins. INGRES employs a dynamic optimization algorithm, while System R* uses a static optimization;
- SDD-1 [BGW+81] was the first query optimization algorithms based on semijoin. It aims to minimize the amount of intersite data transfers through a cost/benefit analysis which sequentially selects the most profitable semijoin to execute; AHY algorithms are a collection of algorithms for minimizing either the response time or the total cost

for a query; W algorithm is a static strategy. Also there are many other algorithms based on semijoin or variant of semijoin;

- Filter-based algorithms are more efficient and popular now. In the next chapter of this report, we will introduce a new filter-based algorithm called Composite Semijoin Filter and compare it to other filter-based algorithm through experimental results.

Query optimization is the important part in distributed database systems. A large number of query optimization algorithms have been proposed by now. But the study in this area is and will be continued.

Chapter 3 Proposed Algorithm

3.1 Problem and Motivation

In distributed database systems, the data is distributed and stored at different sites, which are connected by a computer network. In order to complete a final query, data needs to be transmitted between sites and this communication cost is a dominant factor compared to local processing cost. Because some data are not participating in the final joining query result, it is feasible to discharge them all before transmitting. It is obviously that the cost of transmitting the reduced relations will be lower than that of transmitting the original relations.

The objective of distributed query optimization is to find strategies to minimize the amount of data transmitted over the network. During previous research efforts, semijoin tactics are widely applied for query processing to reduce transmission cost by transmitting only the projections instead of the whole relations. If relations are reduced fully using a semijoin-based algorithm before they are shipped to the join site, less communication cost may be incurred when reduced relations are sent to the result site.

However, due to the type of queries and the independence of attributes assumed in semijoins, the relations appearing in the query may not be reduced fully. As a result, the communication cost in assembling the relation can still be high [YC84]. Sometimes two or more attributes, each with poor selectivity, can be combined to form a composite semijoin with a better selectivity. In this situation it may be beneficial to do the semijoins as one composite rather than as multiple single column semijoins (An example is shown

in figure 2.6). A composite semijoin is a semijoin in which the projection and transmission involve multiple columns. Through simulation results, it has been shown in [PC90] that algorithms including the possibility of composite semijoin can generate strategies which are far better than those that ignore this method. But the cost to transmit composite semijoin projections may be high.

Hash semijoin is proposed to minimize the cost of a semijoin operation (i.e., the cost of transmitting the semijoin projection). It is based on the concept of search filters (also called Bloom filters). A bloom filter is a vector of bits, which is used to filter out the tuples that do not participate in the join. Compared to semijoin, hash semijoin can have lower cost because a Bloom filter is generally smaller than the join attribute projection. Although most research based on filters varies in how the filters are used, the majority encode them using hashing. Hashing is a procedure of applying a special function, called hash function, to a key value to produce an address in a data structure (e.g. a hashed index or a bit array). Unless we have a perfect hash function, filters can never avoid false drops or so called collisions, which occur when two or more attribute values hash to the same address. (The example is shown in figure 2.8.). In [YL99], two Bloom filters are used and the experimental results show that the performance of the algorithm is much better than using a single set of filters under the assumption of collisions. It is impossible to implement composite semijoin using Bloom filters because we cannot keep the relation information between attributes within one tuple when using hash function to hash multi column attributes to one address.

Can we still use filters but avoid collision to get the high performance? The answer is yes. PERF join provides the possibility. It can produce a variation of Bloom filters by scanning the relation tuple order instead of hashing. Hence it does not suffer any loss of join information incurred by hash collisions.

What motivates my interest and future study in this field is how to make improvement based on the current available techniques or algorithms. In this thesis, we take advantage of composite semijoin, Bloom filters and PERF join to propose a new algorithm called Composite Semijoin Filter to implement composite semijoins.

3.2 The Algorithm

In this section, we will introduce our proposed algorithm - Composite Semijoin Filter in detail and give an example to illustrate how this filter works.

Composite Semijoin Filter is a filter-based algorithm, which allows the combination of composite semijoins, Bloom filters and PERF joins. Its primary goal is to reduce the size of relations participating in the final joining, especially to reduce the data that cannot be reduced by using a pure semijoin. As a result, it can minimize the transmitting cost significantly over the network. We use a composite semijoin filter as a reducer.

3.2.1 Description of The Algorithm

The Algorithm can be computed with the following steps. We assume that before using this algorithm, all initial local processing should be done to make sure there are no duplicate records. Each query is processed in two phases.

First reduction phase: (Reduce relations using composite semijoins.)

1. Do all local processing: Do all composite semijoin projections for each relation (Figure 3.1); – *cost1*
2. Creating CSFs for each relation: (Figure 3.2)

    ✓ Send all composite semijoin projections in parallel to the assembling site or other site; – *cost2*

    ✓ Create composite semijoin filters for each relation by scanning the tuple order of common join attributes. – *cost3*

3. Updating filters: If there are more than one CSF for each relation, do "*and*" operation and generate the final filter for this relation, then update other filters related if there is change (Figure3.3); – *cost4*
4. Reducing: (figure3.4)

    ✓ Send CSF back to the site of its relation; – *cost5*

    ✓ Reduce this relation using its own CSF. – *cost6*

*Second reduction phase: (*Reduce relations using hash semijoin.)

Finally, transmitting all reduced relations to the assembling site in parallel to produce the final query result (Figure3.5). – *cost7*

Cost will be incurred during each step. They are represented as *cost1* to *cost7*. The meanings are:

- *Cost1*– the local process cost for projecting the composite semijoin;

- *Cost2*– the communication cost for transmitting composite semijoin projections (the size of projections);
- *Cost3, Cost4* – the local process cost for creating CSF of its relation;
- *Cost5* – the communication cost for transmitting composite semijoin filters (the size of all filters);
- *Cost6* – the local process cost for reducing each relation using CSF;
- *Cost7*– the communication cost for transmitting reduced relation (the size of all reduced relations).

Let us see a simple example to explain how this filter works. Suppose we have three relations, which must be joined to get the query result. There are five join attributes. R_1 has two common join attributes A and B with R_2, one common join attribute B with R_3; R_2 has two common join attributes A and B with R_1, two common join attributes B and D with R_3; R_3 has two common join attributes B and D with R_1. In this example, if we use pure semijoin for R_1 and R_2 on attribute A and B separately, there will be nothing to be reduced. But if we use composite semijoin, the relations will be reduced greatly. So in this situation, it will be more beneficial using our algorithm.

R_1

A	B	C
a	d	g
b	e	h
c	f	i

R_2

A	B	D	E
b	e	c	i
c	d	e	h
a	f	d	g

R_3

B	D	F
d	d	b
e	c	a
f	c	d
g	d	c

Figure 3.1 Original Tables of Relations

First, we do all the composite semijoin projection for R_1, R_2 and R_3. There are two composite semijoin projections for relation R_2: $P_{R2}(A,B)$ and $P_{R2}(B,D)$. The results are shown in Figure 3.2.

$P_{R1}(A,B)$

A	B
a	d
b	e
c	f

$P_{R2}(A,B)$

A	B
b	e
c	d
a	f

$P_{R2}(B,D)$

B	D
e	c
d	e
f	d

$P_{R3}(B,D)$

B	D
d	e
e	c
f	c
g	e

Figure 3.2 Projection of Composite Semijoin

Secondly, send all the projections to a same site (the assemble site or another site) and create CSFs for relation R_1, R_2 and R_3. A CSF is a bit vector that contains one bit for every tuple in P_R and in the same order. So the size of CSF for a relation equals to the number of the relation. Create CSF by scanning tuple order and set the corresponding bit of the filter to 1 if the tuple is matching otherwise to 0.

R_1 has one CSF, R_2 has two CSFs, R_3 has one CSF. So we do "*and*" operation $CSF_{R2}(A,B)$ and $CSF_{R2}(B,D)$ for R2 and get CSF_{R2}. Because there is a change when doing " *and*" operation. So we update related filters ($CSF_{R3}(B,D)$)

Figure 3.3 shows the details.

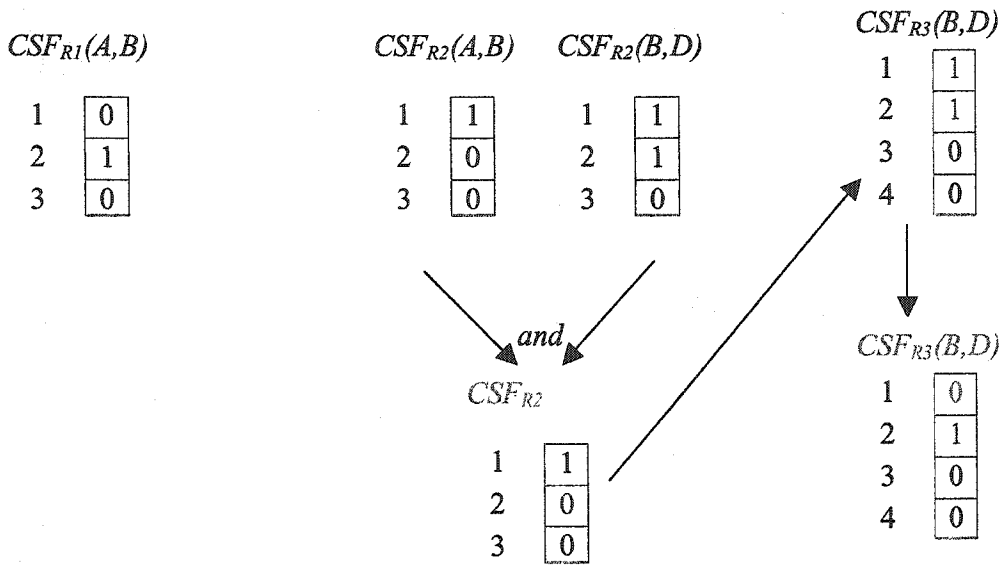


Figure 3.3 Filters of Composite Semijoins

Note: The number left denotes the position of tuples.

Thirdly, transmitting $CSF_{R1}(A,B)$ to site of R_1, CSF_{R2} to site of R_2, $CSF_{R3}(B,D)$ to site of R_3 and reduce R_1 to R_1'using $CSF_{R1}(A,B)$, R_2 to R_2'using CSF_{R2}, R_3 to R_3'using $CSF_{R3}(B,D)$. (Figure 3.4)

Finally, send R_1', R_2' and R3' to the assembling site and compose final result (Figure3.5).

Reducing Relations using their own filter...

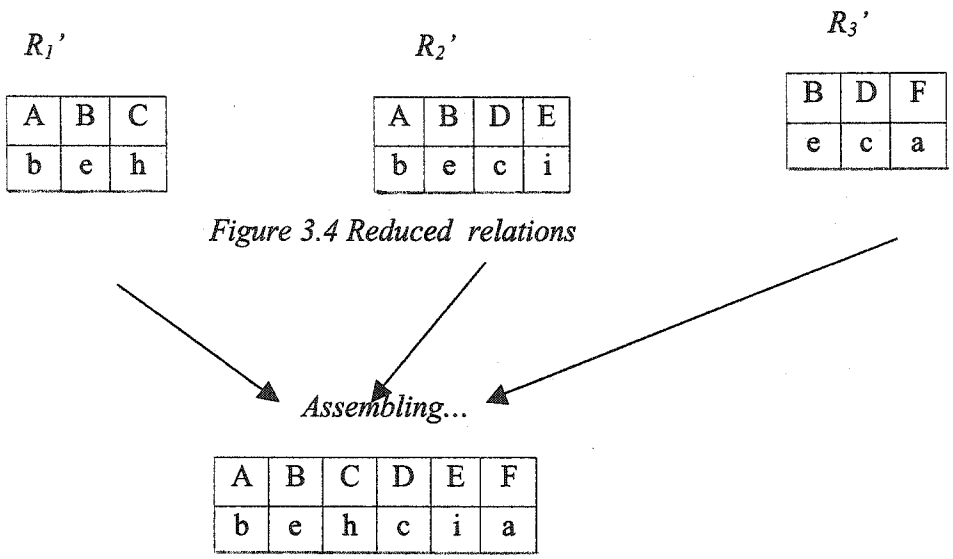


Figure 3.4 Reduced relations

Figure 3.5 Final Result Relation

3.2.2 Implementation

Development environment:

- Microsoft visual C++ 6.0
- Windows XP professional

Main Data Structures:

- **A_matrix[num_rel][num_attr]** -- adjacency matrix, used to show which relations have joining attributes in common.
- Adjacency list: used to represent a query graph. Each relation has one list which head-node contains the number of common attributes, while list-nodes showing connecting relations and common attributes. Each node is defined as:

```
struct rec{
 int attr;                  // attribute id
 int rel;                   // relation id
 struct rec *next;          // pointer to point next element of the list
 };
struct rec * vertex[num_rel]
```

- **csf_final[nul_rel][max_tuple]** – Composite Semojin Filter, use to construct CSF for each relation

Main Functions:

- init_data() – to initialize data
- read_data() – to read data from statistic table
- build_amatrix() – to build adjacency matrix
- build_csf() – to build Composite Semijoin Filter (CSF) for each relation
- reduce_rel() – to reduce each relation using its own CSF
- cal_cost() – to calculate the cost (filter cost and final transmission cost)
- output_file() – to output the result data

Chapter 4 Experiments and Evaluation

In the previous chapter, we introduced our proposed algorithm. Is this algorithm good or not? The experiment is the best way to evaluate it. In this chapter, we will present our experimental scenarios and analyze the experimental results.

4.1 Experimental System

We assume a distributed relational database management system with a number of independent nodes distributed geographically and connected via a point-to-point network. The relations are distributed among the nodes and all nodes can access all data; Each node has local processing and storage capabilities, that means that selections and projections should be carried out during the local processing phase before the applying the algorithm; We will only consider select-project-join (SPJ) queries since most queries can be stated in this format.

The *test-bed* we used is the one developed by the author in [BWT95]. Based on the test database, we construct a set of different queries. Each query consists of an arbitrary number of relations and an arbitrary number of joining attributes. We investigate the following characteristics in this thesis:

- The number of relations involved in the query.
- The number of possible joining attributes involved in a given query.
- The selectivity of the attributes in the query.
- The number of tuples in a relation.
- The domain size of attributes.

4.2 Evaluation Method

The main objectives of evaluating this algorithm (CSF) are to determine how well the algorithm performs. We will measure the performance of the algorithm over Initial Feasible Solution (IFS) in terms of the total cost. IFS ships all relations directly to the query site, where centralized query processing performs joins and builds the query result. The cost of the algorithm includes the cost each projection shipped, the size of filters and the size of reduced relations. We will also compare the algorithm to another filter-based algorithm (W2) in terms of the reduction ratio and the total cost.

4.2.1 Size and Selectivity

For each relation R_i , we use $| R_i |$ to denote the cardinality of R_i, $S(R_i)$ to represent the size of relation R_i in bytes; $W(R_i)$ is the width of a tuple in R_i in bytes. Then:

$$S(R_i) = | R_i | * W(R_i) \qquad (1)$$

The size and selectivity of each individual attribute are represented by $S(d_{ij})$ and $?(d_{ij})$ respectively, the width of the join attribute in bits is $W(d_{ij})$. Then:

$$S(d_{ij}) = | d_{ij} | * W(d_{ij}). \qquad (2)$$

$?(d_{ij})$ is the selectivity on each joining attribute j of relation R_i. It is the number of different values occurring in the attributes divided by the number of all possible values of the attribute. Suppose the cardinality of the joining attribute is $| d_{ij} |$, the domain of d_{ij} is $D(d_{ij})$, the selectivity is commonly defined as

$$?(d_{ij}) = | d_{ij} | / D(d_{ij}) \qquad (3)$$

The selectivity is regarded as high when $?(d_{ij})$ is small. In our experiment, there are three level of selectivity.

Figure 4.1 shows the statistical information of a database including four relations and 2 joining attributes. In this example, the domain of d_{i1} equals $D(d_{i1}) = 990$, the domain of d_{i2} equals $D(d_{i2}) = 610$. The relation R_1 only has one joining attribute 2. The size of R_1 is 4800. The size of the projection of relation R_1 on joining attribute 2 is 867. So we calculate the selectivity of R_1 projected over joining attribute 2 as follow:

$$?(d_{12}) = |d_{12}| / D(d_{12})$$
$$= 435/610$$
$$= 0.713115$$

$S(R_i)$	$S(d_{i1})$	$?(d_{i1})$	$S(d_{i2})$	$?(d_{i2})$
4800	0	0.000000	435	0.713115
1900	945	0.954545	0	0.000000
1700	0	0.000000	525	0.860656
3300	825	0.833333	565	0.926230

Figure 4.1 Database Statistical Information

4.2.2 Cost and Benefit

The total cost of CSF is the sum of the reduced relations and the size of the projection and the size of filters.

$$C(CSF) = ? \; S(R_i') + S(\text{Projections}) + S(\text{Filters}) \quad (i = 1 \ldots n) \qquad (4)$$

The benefit of algorithm CSF is the difference of the size of original relations and reduced relations.

$$B(\mathrm{CSF}) = ?\ (S(R_i) - S(R_i')) \qquad (i=1\ldots n) \qquad (5)$$

The benefit ratio (or reduction ratio) is the benefit over the size of original relations.

$$\mathrm{BR(CSF)} = B(\mathrm{CSF}) / ?\ S(\mathrm{R_i}) * 100 \qquad (i=1\ldots n) \qquad (6)$$

The cost-reduction ratio is the reduced cost over the original cost.

$$\mathrm{CR(CSF)} = (\mathrm{C(IFS)} - C(\mathrm{CSF})) / \mathrm{C(IFS)} * 100 \qquad (i=1\ldots n) \qquad (7)$$

If the benefit exceeds the cost, then the algorithm is called a cost-effective.

Several assumptions are made in our experiment:

- The shipment of one word is a "unit" of cost
- Each attribute value can be represented by one word
- 64-bit word when calculating filter size, so the filter size equals to *max_tuples/64*.

1) Compare to IFS

IFS algorithm is a simple way to process a query. It ships all relations directly to the query site, where centralized query processing performs joins and builds the query result. It is simple but rarely efficient because of the high transmission cost. We compare our algorithm to IFS in order to evaluate the algorithm in terms of total cost.

The total cost of IFS is the sum of the costs of transferring all relations to the joining site. According to our evaluation method, the total cost of IFS should be the sum of the size of all relations participating in the query.

$$C(\text{IFS}) = ?\ S(R_i) \qquad (i = 1 \ldots n)$$

For example, in the example of Figure 4.1, the cost of IFS equals to:

$$4800+1900+1700+3300 = 117000$$

2) Compare to W2 [MO98]

W2 is another algorithm that uses filters. Each filter is an array of bits that functions as a very compact representation of the values of a join attribute in a relation. A perfect hash function is used to set bits in the filter.

The 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. Each relation is usually only processed once. However, if a filter changes then certain relations must be processed again. The algorithm involves two phases.

Phase one: The adjacency list is used to determine the order in which the filters are constructed and used. Repeat all the substeps until each relation has been processed once.

- Select the relation with lowest in_degree for processing
- Scan adjacency list to see which filters must be constructed. If a filter is already available then concurrently use it to reduce the relation and produce all required filters.
- If a filter has changed then use the following "filter rule": if a filter for a relation changes then add that relation to the queue only if it has already

been processed; it is not already on the queue; and it is not the most recently processed the relation.

- 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.

Phase two: The queue is processed in this phase, repeat until the queue is empty.

- Remove relation from the queue.
- Reduce relation using all appropriate filters.
- If a filter changes the use the "filter rule".

Suppose we have three relations which must be joined and shipped to some query site.

R_1

A	B	C
1	2	3
2	3	4
3	4	5

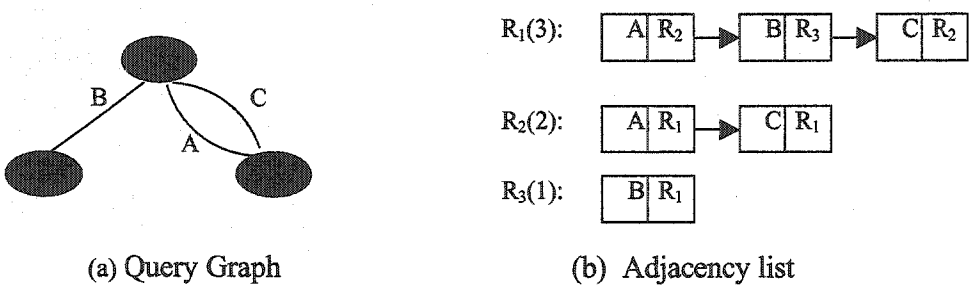
R_2

A	C	D
2	4	5
3	5	6
4	5	7

R_3

B	E
3	4
5	5
6	6

The query graph and adjacency list are represented as follow:



(a) Query Graph (b) Adjacency list

Figure 4.2 An example of algorithm W2

In phase one: First, we select the relation R_3 (with the lowest in_degree) for processing, scan R_3's adjacency list, produce filter for B (3, 5, 6), reduce in_degree of R_1;

Second, select the relation R_1 for processing, scan R_1's adjacency list, filter B is already exist, so reduce R_1 using B, then R_1'={2,3,4}, produce filter A(2), C(4) and new B(3), reduce in_degree R_2 and place R_3 on queue, then mark R_1 as processed;

Third, select R_2 and scan its adjacency list, reduce R_2 using A and C, reduced R_2'={2,4,5}, produce new filter A(2), C(4) and D(5). Because there is no change for filters and all relations have been processed, phase one stops, go phase two.

In phase two, remove R_3 from the queue, reduce R_3 using filter B, then R_3'={3,4}. The queue is empty so the algorithm stops.

All relations are fully reduced:

R_1'

A	B	C
2	3	4

R_2'

A	C	D
2	4	5

R_3'

B	E
3	4

The total cost of W2 is the sum of the reduced relations and the size of filters.

$$C\,(W2) = ?\ S\,(R_i') + S\,(\text{Filters})\ \ (i = 1 \ldots n)$$

The benefit of algorithm W2 is the difference of the size of original relations and reduced relations.

$$B\,(W2) = ?\ \ (S(R_i) - S(R_i'))\qquad (i = 1 \ldots n)$$

4.3 Experimental Results

4.3.1 Experiment Steps

The experimental process includes four steps.

First step:

Create a query by executing the program **create_query.exe.** It should be followed by three parameters:

- *num_rel* - number of relations (3…6)
- *num_attr* - number of attributes (2…4)
- *level_sel* – level of selectivity (0, 1, 2)

Because in practice, the numbers of relations involved in join operations are usually no more than 6, and joining attributes involved are not many. So in this experimental environment, the range for the number of relations is from 3 to 6, while the range for number of joining attributes is from 2 to 4.

Given the desired number of relations and the maximum number of join attributes, the program will produce a query statistics table (Figure 4.1) as well as the input parameters that are required for constructing the actual relations.

The selectivity is classified into 3 categories: 0 represents High selectivity (0.1 – 0.4), 1 represents Medium selectivity (0.4 – 0.7) and 2 represents Low selectivity (0.7 - 0.9).

Second step: Build relations by executing **relbuilder.exe.** *relation_id*" according to the statistical information produced in first step. One parameter is needed.

r*elation_id* : 0...num_rel

Third step: Reduce relations and output the result data by executing **csf_test.exe.**

*Forth step: a*nalyze the experimental results produced in third step.

The query creator (create_query.exe) and relations generator (rebuilder.exe) were created by previous colleagues in Database Group of Windsor University, and revised by me. The CSF reducer (csf_test.exe) is implemented by me.

In our experiment, each relation in the query consists of 500 to 6000 tuples, while the attribute domain contains 500 to 1500 distinct values. Because the number of relations in each query in our experiments is between 3 to 6, the number of attributes varies between 2 to 4, each combination of a relation number and a attribute number can make up a query type. For example query type 4_3 represents four relations and three attributes. In total, twelve query types ranging from 3_2 to 6_4 will be represented in the experiment. Also, the experiments carried out are classified into three parts based on the selectivity of all joining attributes in the test queries. Thirty-six queries were constructed and executed using the algorithm CSF. In order to evaluate this algorithm exactly and effectively, each type of query will be run 60 times (runs). Because in experiments, we found after 50 runs, the results seems no much changes. So over 2160 queries vary in many ways including the number of relations, the number of attributes and the level of selectivity.

4.3.2 Results and Comparison

The objective of our experiment is try to answer the following questions:

1. How does the selectivity of the attributes, the number of relations and the number of joining attributes in the query affect the performance?
2. Does the algorithm perform well compared to other algorithms?

Effects of selectivity and attribute: Figure 4.3 to Figure 4.6 show the effect of selectivity and effect of number of attributes at different number of relations.

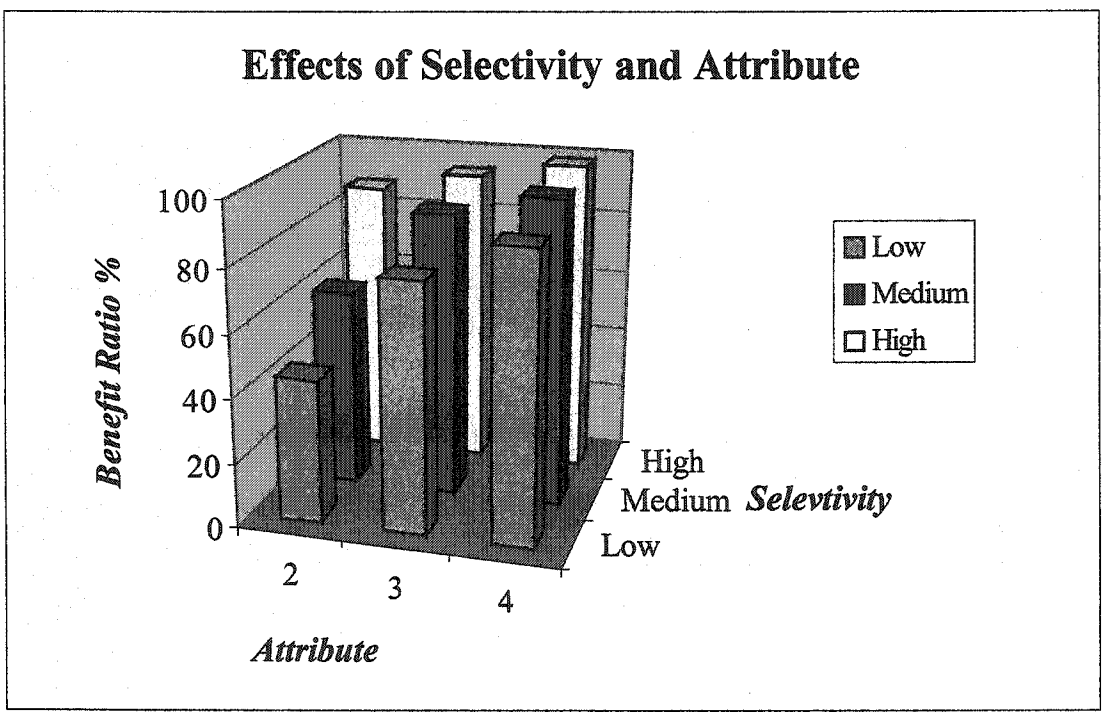


Figure 4.3 Effects of Selectivity and Attributes (Three Relations)

As Figure 4.3 shows, high selectivity always produces a higher benefit ratio than medium or low selectivity in the case of three relations in a query. Under the same selectivity, as the number of attributes increases, the benefit ratio also increases.

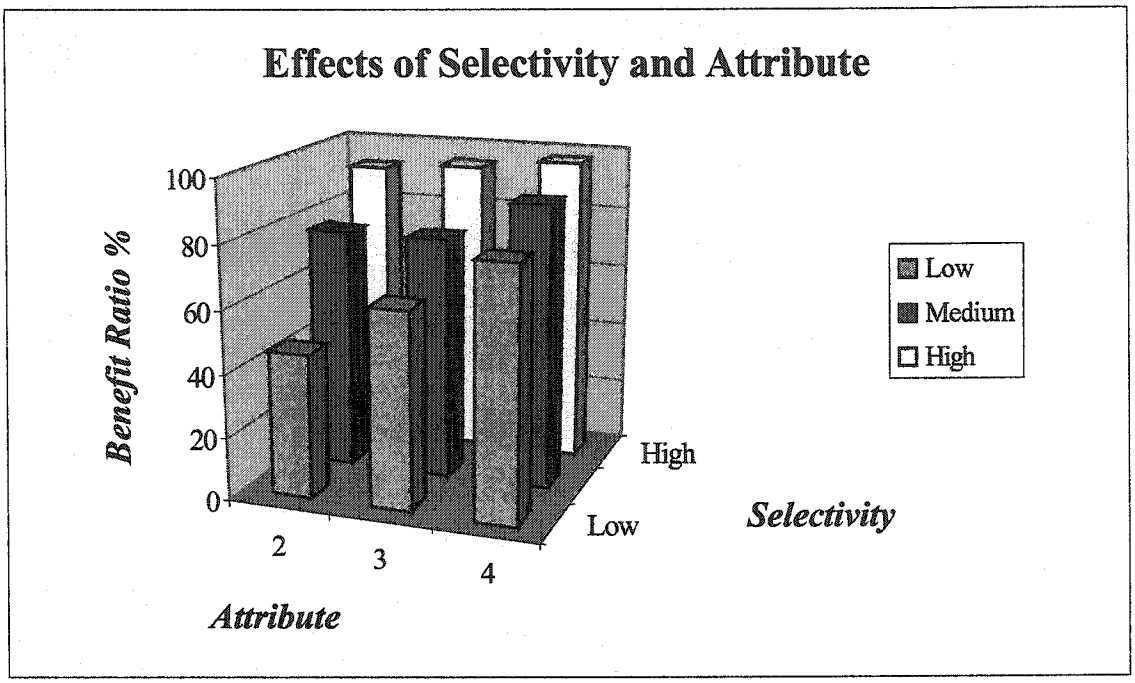


Figure 4.4 Effects of Selectivity and Attributes (Four Relations)

In the Figure 4.4, we can get the same conclusions as the case of three relations.

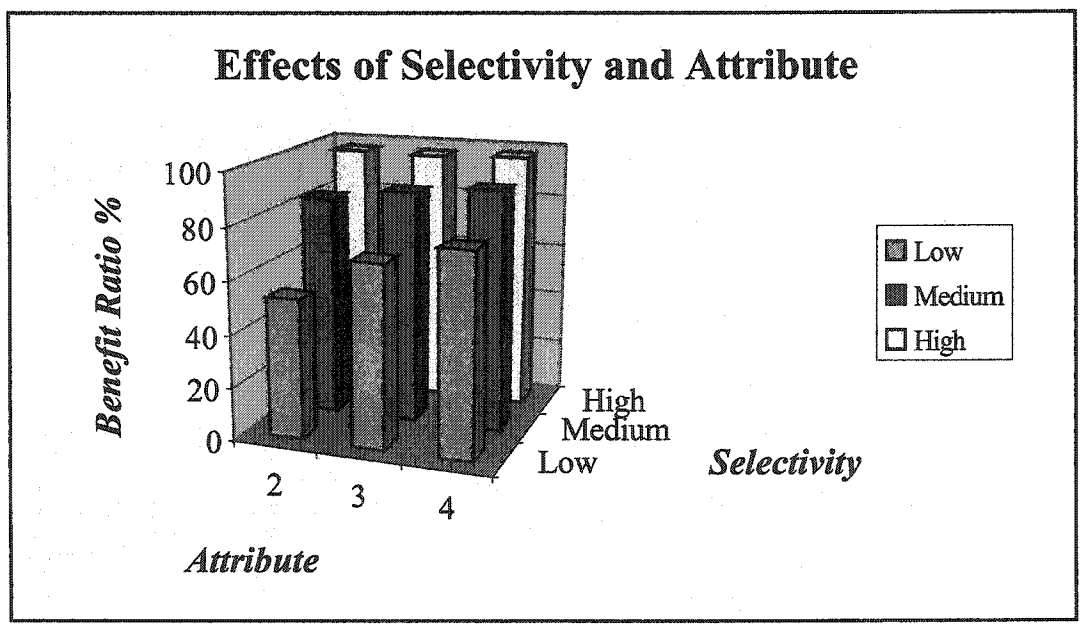


Figure 4.5 Effects of Selectivity and Attributes (Five Relations)

Figure 4.5 shows that when the number of relations increases from 4 to 5, the performance is better under each same case except four attributes with low selectivity.

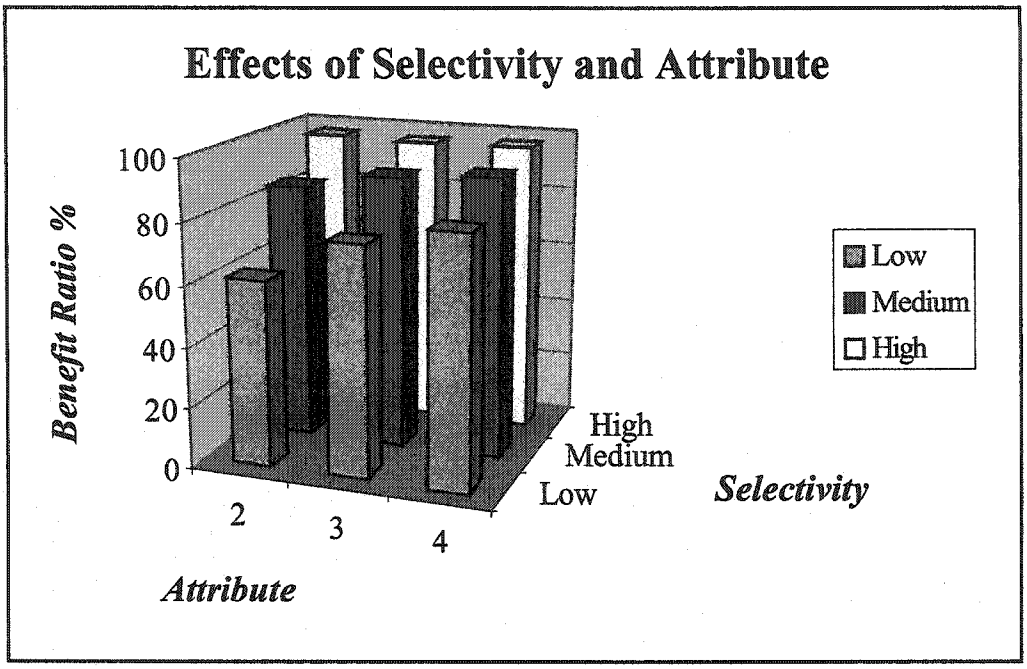


Figure 4.6 Effects of Selectivity and Attributes (Six Relations)

As Figure 4.6 shows, general conclusions got from three relations are also suitable for the case of six relations. Compared to five relations, the performance is better.

General speaking, the higher level of selectivity, the higher beneficial rate produced by the algorithm CSF; With the increasing number of attributes, the benefit ratio also increases; With the same selectivity, when we increase the number of relations from 3 to 6, the beneficial rate also increases a little bit. The exception occurs in the case of three relations with four join attributes.

The Comparison of Reduction Ratio (W2 vs. CSF):

Figure 4.7 shows the comparison results of reduction ratio between algorithm W2 and CSF.

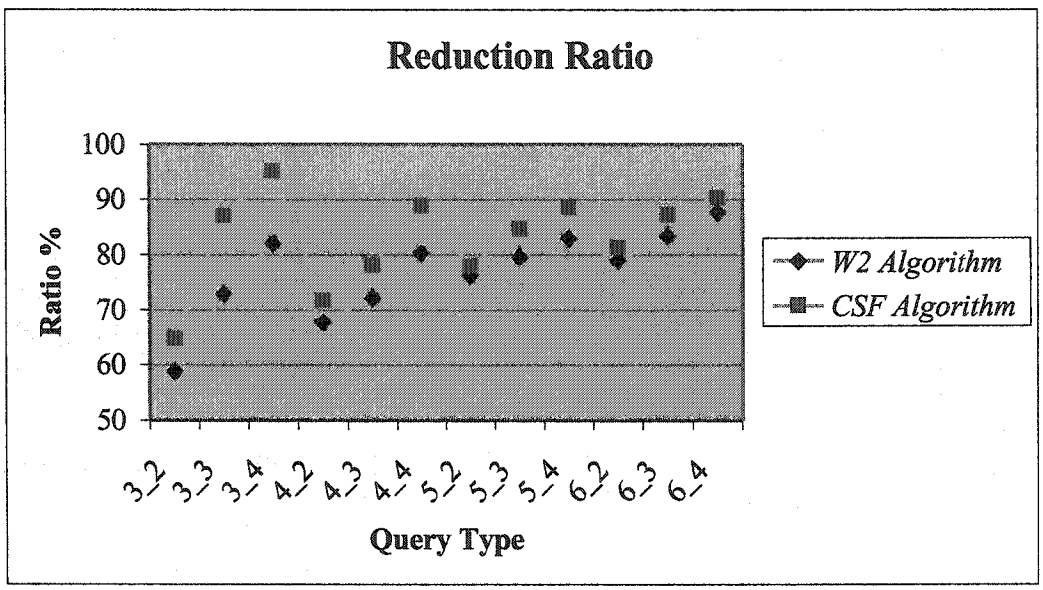


Figure 4.7 Reduction Ratio

As Figure 4.7 shows, the algorithm CSF can always gain more benefit than algorithm W2. With the increasing number of attributes, the algorithm CSF produces higher reduction ratio than that of algorithm W2. That means the algorithm CSF works more efficiently under more common joining attributes. So, algorithm CSF is more beneficial than W2.

The comparison of transmitting cost (IFS, W2 and CSF):

The comparison results of transmitting cost among algorithm IFS, W2 and CSF is shown in Figure 4.8.

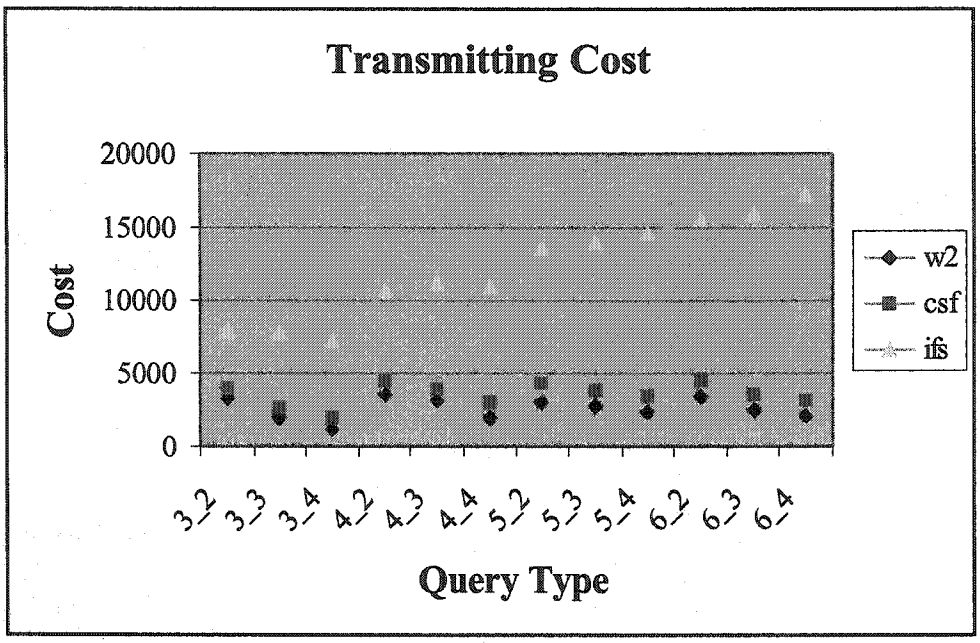


Figure 4.8 Tansmitting cost

As we can see, both algorithm W2 and CSF can reduce transmitting cost significantly. But compared to algorithm W2, CSF produces a little bit more cost. The extra cost is caused during transmitting composite semijoin projections when applying the algorithm CSF. And when the number of relations and the number of attributes are all the biggest, the transmitting cost will be reduced most by using the algorithm CSF or W2.

Summary:

Figure 4.9 gives the tables of comparison between W2 and CSF in Benefit Ratio.

Beneficial Ratio(%)		Number of Join Relations											
		3						4					
		High		Med		Low		High		Med		Low	
		W2	CSF	W2	CSF	W2	CSF	W2	CSF	W2	CSF	W2	CSF
Number of Attributes	2	86.77	87.89	59.25	61.90	30.31	44.71	90.47	92.46	76.21	76.93	36.49	45.76
	3	93.59	93.90	79.98	89.31	45.28	78.05	93.26	94.58	75.56	77.01	47.69	62.69
	4	98.59	98.92	90.34	96.45	57.16	90.07	96.04	97.27	88.50	89.93	56.30	79.55
Average		92.98	93.57	76.52	82.55	44.25	70.94	93.26	94.77	80.09	81.29	46.83	62.67
Beneficial Ratio(%)		5						6					
		High		Med		Low		High		Med		Low	
		W2	CSF	W2	CSF	W2	CSF	W2	CSF	W2	CSF	W2	CSF
Number of Attributes	2	96.02	97.23	81.27	83.46	51.42	52.99	96.39	97.37	83.31	84.94	57.30	61.34
	3	94.75	96.83	84.13	88.18	59.82	69.31	91.13	96.03	87.00	90.27	71.98	75.63
	4	96.88	97.94	89.21	91.07	62.81	76.69	96.39	97.05	90.87	92.45	75.77	81.95
Average		95.88	97.33	84.87	87.57	58.02	66.33	94.64	96.82	87.06	89.22	68.35	72.97

Figure 4.9 Table of Benefit Ratio

From the data in the table, we can say that the algorithm CSF is always beneficial. The lowest benefit ratio is 44.71% (three relation two join attributes at low selectivity), and highest one is 98.92% (three relation four attributes at high selectivity). The average is 94.72% at high selectivity, 85.16% at medium selectivity and 68.23% at low selectivity. In general, the average of benefit ratio is 82.70% compared to the algorithm IFS. CSF can get higher benefit ratio (5.80%) than algorithm W2.

The table of the comparison between W2 and CSF in terms of Cost-Reduction Ratio is shown in Figure 4.10.

Cost-reduction Ratio(%)		Number of Join Relations											
		3						4					
		High		Med		Low		High		Med		Low	
		W2	CSF	W2	CSF	W2	CSF	W2	CSF	W2	CSF	W2	CSF
Number of Attributes	2	86.95	77.91	58.77	47.54	30.01	24.90	92.12	83.14	76.01	63.35	36.13	31.52
	3	93.32	84.25	79.19	74.11	54.18	43.52	93.88	85.63	77.12	64.95	46.66	45.69
	4	98.51	85.90	89.98	80.49	65.53	55.78	96.95	87.97	88.14	78.33	62.65	54.53
Average		92.92	82.69	75.98	67.38	49.91	41.40	94.31	85.58	80.43	68.88	48.48	43.91
Cost-reduction Ratio(%)		5						6					
		High		Med		Low		High		Med		Low	
		W2	CSF	W2	CSF	W2	CSF	W2	CSF	W2	CSF	W2	CSF
Number of Attributes	2	97.10	90.41	83.34	72.82	51.29	40.21	95.66	92.51	84.26	74.10	55.73	48.91
	3	96.69	88.95	88.32	74.57	58.83	57.09	90.53	91.02	90.80	78.51	71.58	64.38
	4	97.71	90.82	90.68	79.86	65.84	63.04	96.53	92.18	91.05	82.57	75.71	70.61
Average		97.17	90.06	87.44	75.75	58.65	53.45	94.24	91.90	88.70	78.39	67.67	61.30

Figure 4.10 Table of Cost-Reduction Ratio

From the table, we can get that the data transmission cost of both W2 and CSF is reduced significantly compared to the IFS. The cost-reduction ratio of CSF is between 24.90% (three relations two join attributes at low selectivity) and is 92.51% (six relations two attributes at high selectivity). The average is 87.56% at high selectivity, 72.6% at medium selectivity and 50.02% at low selectivity. In general, the cost can be reduced 70.06% for average. Compare to algorithm W2, CSF causes a little bit more transmission cost (13.04%).

Figure 4.11 is the table of benefit, cost and net benefit of Algorithm CSF. In most cases, CSF has been shown a cost effective algorithm (Net-benefit is positive.). Exceptions occur when selectivity is low. With the low selectivity, the relations will be reduced less so the cost will be higher than medium or high selevtivity.

		Number of Join Relations								
		3								
		High			Medium			Low		
		Benefit	Cost	Net-Bef	Benefit	Cost	Net-Bef	Benefit	Cost	Net-Bef
Num_attr	2	5825	1652.13	4172.87	3892	4295.05	403.05	1991	6004.99	-4013.99
	3	6372	1191.04	5180.96	5839	2040.01	3798.99	3449	4476.34	-1027.34
	4	5267	864.74	4402.26	6373	1545.00	4828.00	4306	3413.84	892.16
Average		5821.33		4585.36	5368.00	2626.68	3010.01	3248.67	4631.72	-1383.06
		4								
		High			Medium			Low		
		Benefit	Cost	Net-Bef	Benefit	Cost	Net-Bef	Benefit	Cost	Net-Bef
Num_attr	2	8109	1644.69	6464.31	6892	3987.58	2904.42	3517	7644.06	-4127.06
	3	9052	1519.12	7532.88	7558	4079.43	3478.57	5213	6198.41	-985.41
	4	8435	1154.05	7280.95	9039	2500.89	6538.11	6389	5328.26	1060.74
Average		8532.00	1439.29	7092.71	7829.67	3522.63	4307.03	5039.67	6390.25	-1350.58
		5								
		High			Medium			Low		
		Benefit	Cost	Net-Bef	Benefit	Cost	Net-Bef	Benefit	Cost	Net-Bef
Num_attr	2	11906	1263.27	10642.73	10427	3892.47	6534.53	5361	7972.57	-2611.57
	3	11515	1430.97	10084.03	10214	3483.32	6730.68	8762	6585.72	2176.28
	4	12087	1222.42	10864.58	11586	2921.68	8664.32	10194	5975.78	4218.22
Average		11836.00	1305.56	10530.44	10742.33	3432.49	7309.85	8105.67	6844.69	1260.98
		6								
		High			Medium			Low		
		Benefit	Cost	Net-Bef	Benefit	Cost	Net-Bef	Benefit	Cost	Net-Bef
Num_attr	2	14111	1141.83	12969.17	11199	3913.58	7285.42	8022	8380.38	-358.38
	3	13891	1371.24	12519.76	12625	3455.94	9169.06	10693	5915.90	4777.10
	4	15445	1310.62	14134.38	14097	2976.06	11120.94	12505	5206.01	7298.99
Average		14482.33	1274.56	13207.77	12640.33	3448.53	9191.81	10406.67	6500.76	3905.91

Figure 4.11 Table of benefit, cost and net-benefit of Algorithm CSF

Chapter 5 Conclusions and Future Work

5.1 Conclusions

Query optimization is an important part in distributed database systems. The main concern in this area is the selection of the best sequence of various operations to process queries to minimize the communication cost. Because finding the optimal solution is NP-hard, heuristics are applied to find near-optimal processing strategies.

During the past two decades, various possible algorithms have been presented and tested, which can be classified into following categories: Join-based algorithms, Semijoin-based algorithms, Filter-based algorithms, and join/semijoin combined algorithms.

Semijoin is often a common starting point for join algorithm in distributed database. It is widely used to reduce the amount of data transferred between sites. Semijoin-based algorithms perform better than join-based algorithms. However, we still have to spend a lot for transmitting the semijoin projection when using semijoin-based algorithms. Also in most of the algorithms, multiple semijoins may be performed with common source and common result sites. In this situation it may be beneficial to do the semijoins as one composite rather than as multiple single column semijoins. But composite semijoin may produce more cost than semijoin when transferring composite semijoin projection.

Filters are proposed as a cheap way to minimize this cost by transmitting filters instead of projections. However, since Bloom filters are constructed by hash functions, collisions can never be avoided. This is the problem or bottleneck for Bloom filter-based

algorithms. PERF join provides a new idea of creating filters to overcome the problem by scanning the tuples and reserving the poison information instead of hashing the values.

In this thesis, we take the idea of the PERF join and create a variation of Bloom filter called Composite Semijoin Filter (CSF) to implement composite semijoin and avoid collisions. The algorithm can process general queries consisting of an arbitrary number of relations and joining attributes.

The primary goal of our algorithm is to minimize transmitting cost, which is spent during transmitting relations to the assembling site for final query processing. It is implemented by reducing relation size. We use composite semijoins to reduce relations, especially the tuples that may not be reduced by pure semijoins. The second goal is to minimize intermediate transmission cost (caused by transmitting composite semijoin projections) by using a variation of Bloom filter that can avoid collisions.

Although a composite semijoin itself may not be beneficial because of its more total time cost, it always gainful to the execution of subsequent join operations. Our proposed algorithm is evaluated and compared with initial feasible solution (IFS) and another filter-based algorithm in terms of the total cost. From the experimental results, we get the conclusions as follows:

1. The algorithm is always beneficial compared to the IFS. The lowest benefit ratio is 44.71% (three relation two join attributes at low selectivity), and highest one is 98.92% (three relation four attributes at high selectivity). The average of benefit ratio is 82.70% compared to the algorithm IFS.

2. The data transmission cost is reduced significantly compared to the IFS. The cost can be reduced 70.06% for average.
3. Generally speaking, this algorithm is cost-effective (Net-benefit is positive.).
4. The number of relations and the level of selectivity are two main factors which affect the benefit ratio and the cost-reduction ratio.
    - With the same number of relations, no matter how many attributes the relations have, the higher the selectivity, the higher benefit ratio and cost-reduction ratio.
    - With the same selectivity and a fixed number of relations, the more attributes the relations have, the more benefit will gain, and so does the cost reduction ratio.
    - With the same selectivity and the number of attributes, as the number of relations involved in the query increases, applying the algorithm CSF will get more benefit. (Exceptions occur when the query has three relations with three or four joining attributes.)
5. Compare to another filter-based algorithm (W2), CSF can also get higher benefit ratio (5.80%) but will cause a little bit more transmission cost (13.04%).

Consequently, the algorithm proposed in this thesis performs well. The data transmitting cost is reduced significantly in comparison to the algorithm IFS. Compared to the algorithm W2, our proposed algorithm can also produce higher benefit ratio, but will spend more by transmitting the projections. This is the main disadvantage of the algorithm.

5.2 Future Work

Here are several things which should be done but have not been considered in this thesis.

The response time: As we introduced in Chapter two, there are two cost models, one is the response time, and the other is the total cost. In my current work, I just evaluate the algorithm in terms of the total cost. So what we will do later is to evaluate if this algorithm can also get lower response time compared to other algorithms.

The local processing time: In this thesis, we do not take the local processing time into account, but it should not be ignored, for example, the cost for processing the projections, the cost for removing duplicates and the cost for generating the filters. With the augmentation of the relation size, the cost will be larger. So, the total cost will be higher in practice.

The duplicates: If we can find out an efficient method to deal with the duplicates on projections, the transmission cost will be reduced more.

Special cases for composite semijoins: Obviously, the algorithm will perform best if it can be used under special cases which are suitable for composite semijoins.

More runs: Some exceptions occur in the experiments. If we can run more times such as hundreds or thousands times for each query type, the results may be smoothly.

The real system: If possible, it should be put into the real distributed database systems for performance testing and improvement in the future.

References

[AHY83] P. Apers, A. Hevner and S. Yao, "Optimization algorithms for distributed queries", IEEE Transactions on Software Engineering, 9(1), pp.51-60, 1983.

[AM91] J. Ahn, 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. Bernstein, D. M. Chiu, "Using semijoins to solve relational queries", JACM, Vol. 28, No. 4, pp.25-40, Nov. 1881.

[BFS00] S. Bandyopadhyay, Q. Fu and A. Sengupta, "A Cyclic multi-relation semijoin operation for query optimization in distributed databases", IEEE, 2000.

[BG81] P. A. Bernstein, N. Goodman, "The power of natural semijoins", SIAMJ. Computer, Vol. 10. 4, pp.751-771, Nov. 1881.

[BG93] D. Bell, J. Grimson, "Distributed Database Systems", Addition-Wesley.

[BGW+81] P.Bernstein, N. Goodman, E. Wong, C. Reeve, and J.Rothie, "Query processing in a system for distributed databases(SDD-1)", ACM Transactions on database Systems,Vol.6(4), pp.602-625, 1981.

[BKK+01] R. Braumandl, M. Keidl, A. Kemper, D.Kossmann, S.Seltzsam,K. Stocker, "Object Globe: Open Distributed Query Processing Services on the internet", IEEE Computer Society Technical Committee on data Engineering, pp.1-7, 2001.

[BL82] P. A. Black,W. S. Luk, "A new heuristic for generating semi-join programs for distributed query proceeding", In Proc. IEEEE COMPSAC, pp.581-558, Dec. 1982

[BPR90] P. Bodorik, J. Pyra and J. S. Riordon, "Correcting execution of distributed queries", In Proc. Of 2nd int. Symp. on Databases in Parallel and distributed Systems, pp.192-201, 1990.

[BR88] P. Bodorik, J.S. Riordon, "Distributed query processing optimization objectives", Fourth International Conference on Data Engineering, pp. 320 – 329, 1988.

[BR88] P. Bodorik, J. S. Riordon, "Heuristic Algorithms for Distributed Query Processing", IEEE, pp.144-155,1988.

[BRB+01] C. Badue, B. Ribeiro-Neto, R. Baeza-Yates, N. Ziviani, "Distributed query processing using partitioned inverted files", String Processing and Information Retrieval, 2001. Proceedings. Eighth International Symposium on , 2001, pp. 10 - 20.

[BRJ89] Peter Bodorik, J. Spruce Riordon and C. Jacob, "Dynamic Distributed Query Processing Techniques", ACM, pp.349-357, 1989.

[BRP92] Peter Bodorik,J. Spruce Riordon and James S. Pyra, "Deciding to Correct Distributed Query Processing", IEEE Transactions on Knowledge and data Engineer in Vol.4 No.3, pp.253-265, Jun. 1992.

[CBH84] D. M. Chiu, P. A. Berstain and Y. C. Ho, "Optimizing chain queries in a distributed database system", SIAMJ. Computer, Vol. 13. No.1, pp.116-134,1984.

[CCY92] Tung-Shou Chen, Arbee L.P. Chen and Wei Pang Yang, "Hash-semijoin: A new technique to minimizing Distributed Query time", IEEE, 1992.

[CH82] W.W. Chu, P. Hurley, "Optimal Query Processing for Distributed Database System", IEEE. Trans. On Comput., Vol. c-31,no.9, pp.135-150, Sep. 1982.

[CH84] D. M. Chiu, Y. C. Ho, "Optimizing star queries in a distributed database system: A method for interpreting tree queries into optimal semijoin expressions", VLDB, pp.959-967,1984.

[Cha82] Jo-Mei Chang, "A Heuristic Approach to Distributed Query Processing", Proceeding of the 8th VLDB Conference, pp.54-61, 1982.

[Che90] Arbee L. P. Chen, "Outerjoin optimization in multi-database systems", Proceedings of the second international symposium on Databases in parallel and distributed systems, pp.211-218, July 1990.

[CL00] H. Chen and C. Liu, "An efficient algorithm for processing distributed queries using partition dependency", Parallel and Distributed Systems, pp. 339 –346, 2000.

[CL84] L.Chen and V.Li, "Improvement algorithms for semi-jion query processing programs in distributed database system", IEEE Transaction on computers, Vol. 33(11), pp.959-967, 1984.

[CL84] L.Chen and V.Li, "Optimizing star queries in a distributed database system", Proc. 10th int. Conf. Very Large Data Base, pp.136-145,1984.

[CL85] A. L. P. Chen, V. O. K. Li, "An optimal algorithm for distributed star queries", IEEE Trans. On Software Engineering, Vol.11 No. 10, pp.1097-1107,1985.

[CL90] L. chen and V.Li, "Domain-specific semi-join: A new operation for distributed query processing", Information science,Vol. 52, pp.165-183, 1990.

[CY90] M.-S. Chen, P.S. Yu, "Using combination of join and semijoin operations for distributed query processing", Distributed Computing Systems, Proceedings., 10th International Conference on, pp. 328 –335,1990.

[CY90] Ming-Syan Chen and Philip S. Yu, "Using join operation as reducers in distributed query processing", IBM Research report RC15107, pp.116-123,1990.

[CY91] Ming-Syan Chen and Philip S. Yu, "Determing Beneficial Semijoins for a join Sequence in Distributed Query Processing", IEEE, pp.50-58, 1991.

[CY92] M.S. Chen, P.S. Yu, "Interleaving a Join Sequence with Semijoins in Distributed Query", Parallel and Distributed Systems, IEEE Transactions on, Volume: 3 Issue: 5, pp.611 –621, Sept. 1992.

[CY93] Ming-Syan Chen and Philip S. Yu, "Combining Jion and Semi-Jion Operations for Distributed Query processing", IEEE Transactions and Data Engineering Vol.5, No.3, pp.534-542, 1993.

[CY94] Ming-Syan Chen, P.S. Yu, "A graph theoretical approach to determine a join reducer sequence in distributed query processing", Knowledge and Data Engineering, IEEE Transactions on, Volume: 6 Issue: 1, 152 –165, Feb. 1994.

[CY96] Ming-Syan Chen, Philip S. Yu, "Optimization of parallel execution for multi-join queries", IEEE, pp.416-428, 1996.

[ESW78] R. Epstein, M. StoneBraker, and E. Wong, "Query Processing in a Distributed Relational Database System", In Proc. ACM SIGMOD Int. Conf. On Management of Data, pp.169-180, May 1978.

[ES80] R. Epstein and M. StoneBraker, "Analysis of distributed database processing strategies", In Proc. 6th Int. Conf. On Very Large Data Bases,1980

[Fre89] J. C. Freytag, "The Basic Principles of Query Optimization in Relational Database Management Systems", Information Processing 89, G. X. Ritter, Elsevier Science Publishers B. V. (North-Holland), pp.801-807,1989.

[GM95] Bojan Groselj and Wutaibah M. Malluhi, "Combinatorial optimization of Distributed Queries", IEEE Transactions on Knowledge and data Engineering Vol.7 NO.6, pp. 915 –927, 1995.

[Gra96] Jim Gray, "Data Management: Past, Present, and Future", Microsoft Research, Technical Report MSR-TR-96-18, Jun. 1996.

[Gre98] Michael Gregory, "Genetic Algorithm Optimization of Distributed Database Queries", IEEE, pp.271-267, 1998.

[GS86] Bezalel Gavish and Arie Segev, "Set query optimization in distributed database systems. ACM Trandactions on Database systems Vol. 11 No. 3, pp. 265-293,1986.

[GW89] G. Graefe and K. ward, "Dynamic Query evaluation plans", ACM SIGMOD, pp.73-170, 1989.

[HCY94] Hui-I Hsiao, Ming-Syan Chen and Phillip S.Yu, "On parallel execution of multiple pipelined hash joins", ACM SIGMOD, pp. 185-196,1994.

[HCY97] Hui-I Hsiao, Ming-Syan Chen and Phillip S.Yu, "Parallel Execution of Hash joins in Parallel Database", IEEE, pp.872-883, 1997.

[HF00] Ramzi Haraty and Roula Fany, "Distributed query optimization using perf Jions", ACM, pp.284-287, 2000.

[HKL94] Jorng-Tzong Horng, Cheng-Yan Kao and Baw-Jhiune Liu, "A genetic Algorithm for database query optimization", IEEE, 1994.

[HM00] Abdelkader Hameurlain and Franck Morvan, "An Overview of Parallel Query Optimization in Relational Systems", IEEE,2000.

[HWY85] Alan R. Hevner, O. Qi Wu and S. Bing Yao, "Query Optimization on Local Area Networks", ACM transaction on office Information Vol.3, No. 1, pp.35-62,1985.

[HY79] A.R. Hevner and S. B. Yao, "Query Processing in Distributed Databases", IEEE Trans. on Software Eng., SE-5(3), pp.1770-187, 1979.

[JK84] Matthias Jarke and JurGen Koch, "Query Optimization in database system", ACM Computing Surveys Vol.16, No.2, pp.112-152,1984.

[JPS93] Anant Jhingran, Sriram Padmanabhan and Ambuj Shatdal, "Join Query Optimization in Parallel Database Systems", IEEE, 1993.

[KHY82] Y. Kambayashi, M. Yoshikawa and S. Yajima, "QUERY Processing for Distributed Databases Using Generalized Semijoins", ACM Proceedings of SIGMOD, pp.151-160,1982.

[Kos00] Donald Kossmannn, "Cache Investment: Integrating Query Optimization and distributed Data placement", ACM Transaction on Database Systems Vol. 25,No.4, pp.517-558, Dec. 2000.

[Kos00] Donald Kossmannn, "The State of the Art in Distributed Query Processing", ACM Computing Surveys (CSUR), v.32 n.4, pp.422-469, Dec. 2000.

[Kos98] Donald Kossmann, "Iterative Dynamic Programming: A New Class of Query Optimization Algorithms", ACM Transactions on Database Systems,1998.

[KR87] H. Kang and N. Roussopoulos, "Combining joins and Semijoins in Distributed Query processing", Tech. Rep. CS-TR-1794,Univ. Maryland, College Park,1987.

[KR87] H. Kang and N. Roussopoulos, "Using 2-way Semijoins in Distributed Query Processing", IEEE. Proc. of 3rd Int'l Conf on Data Engineering, pp.644-650,1987.

[KS00] Donald Kossman and Konrad Stoker, "Iterative Dynamic Programming A new Class of Query Optimization Algorithm," ACM Transaction on Database Systems, Vol.25, No.1, pp.43-82, 1986.

[LC01] Chang-Hung Le and Ming-Syan Chen, "Distributed query processing in the Internet: exploring relation replication and network characteristics", Distributed Computing Systems, 21st International Conference on. , Apr 2001, pp. 439 - 446.

[LCK96] Chenwen Liu, Hao Chen and Warren Krueger, "A distributed Query Processing Strategt using Placement Dependency", IEEE, pp.477-484,1996.

[Lia98] Yan Liang, "Reduction of collisions in bloom filters during distributed query optimization", M.sc Theses, Computer Science, University of Windsor,1998.

[LOZ95] Xuemin Lin, Maria E. O. and Xiaofang Zhou, "Using Parallel Semi-jion Reduction to minimize Distributed Query response time", IEEE,1995.

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

[LR95] Zhe Li and Kenneth A. Ross, "PERF Join: An Alternative To Two-way Semijoin And Bloomjoin", 1995.

[LY93] Chenwen Liu and Clement Yu, "Performance issues in distributed Query Processing. IEEE, pp.889-905,1993.

[Ma99] Xiaobo Ma, "The use of bloom filters to minimize response time in distributed query processing", M.sc Theses, Computer Science, University of Windsor,1999.

[MB95] J. Morrissey and S. Bandyophdhyay, "Computer Communication Technology and its effects on Distributed Query Optimization Strategies", IEEE, pp.598-601,1995.

[MB96] JM. Morrissey and WT. Bealor, "Minimizing data transfers in distributed query processing: a comparative study and evaluation", The computer journal, Vol.39, No. 8, pp. 675 – 687, 1996.

[Mit02] Michael Mitzenmacher, "Compressed Bloom Filter", IEEE, pp.604-612, 2000.

[MO97] J. Morrissey and W.K. Osborn, "Experiences with the use of reduction filters in distributed query optimization", In proceedings of the 9th international Conference On Parallel and Distributed Computing and Systems (PDCS'97), pp.327-330, 1997.

[MO98] J. Morrissey and W.K. Osborn, "Distributed Query Optimization using reduction filter", Electrical and Computer Engineering, 1998. IEEE Canadian Conference on, Volume: 2, pp.707-710,1998.

[MO99] J.Morrissey and O.Ogunnbadejo, "Combining semijoins and hash-semijoins in a distributed query processing strategy", Electrical and Computer Engineering, 1999 IEEE Canadian Conference on, Volume: 1, pp.122 – 126,1999.

[MO99] J.M. Morrissey, W. K. Osborn, "The effect of collisions on the performance of reduction filter", Proceedings of the 1999 IEEE Canadian Conference n electrical and computer Engineering, pp.215-219, 1999.

[MOL00] J.Morrissey,Wendy Osborn and Y Liang, "Collisions & reduction filters in distributed query processing", Conadian Conf. Electr. Computing Eng. IEEE, pp.240-244, 2000 Vol.1.

[Mor96] J.M. Morrissey, "Reduction filters for minimizing data transfers in distributed query optimization", IEEE, pp.198-201,1996.

[Mul90] James K. Mullin, "Optimal Semijoins for Distributed Database System", IEEE Transactions on Software Engineering Vol.16 No.5, pp.558-560, 5/1/1990.

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

[Mul94] J.K. Mullin, "Optimal Semijoins for Distributed Database Systems", Software Engineering, IEEE Transactions on, Volume: 16 Issue: 5, pp. 558 – 560,1994.

[Mul96] Craig S. Mullins, "Distributed Query Optimization", Technical Enterprise, 1996.

[NS98] F. Najjar and Y. Slimani, "Distributed optimization of cyclic queries with parallel semijoins", IEEE, pp. 717 -722,1998.

[Osb98] W. Osborn, "The use of reduction filters in distributed query optimization", Master's Thesis, the university of Windsor,1998.

[OV91] M. Tamer Ozsu and Patrick Valduriez, "Distributed Database Systems: Where are we now?", ACM Computing Survey, pp.68-78, Aug. 1991.

[OZSU99] M. Tamer Ozsu, "Principles of Distributed Database Systems", Printice Hall, Upper saddle River, New Jersey 07458, Second Edition (1999).

[PC90] William Perrizo, Chun-Shwu Chen, " Composite Semijoins in Distributed Query Processing", Information Sciences Vol. 50,No.3, pp.197-218,1990.

[Per85] W. K. Perrizo, "Upper bound response time semijoin strategies", 1st international conference on super computer systems, pp.273-279,1985.

[PLO91] Hwee Hwa Pang, HongJun Lu and BengChin Ooi. An Efficient Semantic Query Optimization Algorithm. IEEE, pp.326-335,1991.

[PRW94] William Perrizo, Prabhu Ram and David Wenberg, "Distributed Jion Processing Performance Evaluation", Proceeding of the Twenty-Seventh Annual Hawaii International Conference on System Sciences, 1994.

[PV88] Sakti Pramanik and David Vineyard, "Optimizing join queries in distributed database", IEEE Transaction on software engineering Vol. 14, No. 9, pp1319-1326, Sept. 1988.

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

[SB82] M.S. Sacco and S.B. Yao, "Query Optimization in Distributed Database System", In M. C. Yovits (ed.), Advances in Computers,Volum21,New York: Academic press, pp225-273,1982.

[Seg86] A. Segev, "Global Heuristic for Distributed Query Optimization", Proceedings of IEEE INFOCOM, pp.388-394, 1986.

[SE97] Peter Scheuermann and Eugene Inseok Chong, "Adaptive algorithms for join processing in distributed database systems", Distributed and Parallel Databases Vol. 5,no.3, pp. 233-269, Jul. 1997.

[SKB+00] Konrad Stocker, Donald Kossmann, Reinhard Braumandl, Alfons Kemper, "Integrating Semi-Join-Reducers into State-of-the-Art Query Processors", ICDE, 2000.

[SMK97] Michael Steinbrunn, Guido Moerkotte and Alfons Kemper, "Heuristic and Randomized Optimization for the Join Ordering Problem", VLDB Journal: Very Large Data Bases, 1997.

[SW91] Dennis Shasha and Tsong-Li Wang, "Optimizing Equijion Queries in Distributed Databases Where Relations are Hash Partitioned", ACM Transactions on Database Systems, Vol.16 No.2, pp.279-308,1991.

[SSL+02] S.Y. Sung, Peng Sun, Zhao Li and C.L. Tan, "Virtual-join: a query execution technique performance", Computing and Communications Conference, 21st IEEE International , 2002, pp.353 – 357.

[TC02] P.S.M. Tsai, A.L.P. Chen, "Optimizing queries with foreign functions in a distributed environment", Knowledge and Data Engineering, IEEE Transactions on, Volume: 14 Issue: 4, pp. 809- 824, 2002.

[TC92] Judy C.R.Tseng and Arbee L.P. Chen, "Improving Distributed Query Processing by Hash-Semijion", Journal of information Science and engineering, pp.525-540, 1992.

[Vla97] Richard Vlach, "Query Processing in Distributed Database System", 1997

[Wan90] Chihping Wang, "The complexity of processing tree queries in distributed databases", IEEE, pp.604-601,1990.

[WC96] Chihping Wang and Ming-Syan Chen, "On the Complexity of Distributed Query Optimization", Knowledge and Data Engineering, IEEE Transactions on, Volume: 8 Issue: 4, pp.650-662,1996.

[WCS92] Chihping Wang, Arbee L. P. Chen and Shiow- Chen Shyu, "A parallel execution method for minimizing distributed Query response time", IEEE transactions on Parallel and distributed Systems, 3(3), pp.325-333,1992.

[WLC91] Chihping Wang, Victor OK Li and Arbee LP Chen, "Distributed Query Optimization by One-Shot Fixed-Precision Semi-Join Execution", Processing 7th International Conference on Data Engineering, pp.756-763,1991.

[Won77] E. Wong, "Rtrieving dispersed Data from SDD-1", Proc. 2nd Berkely Workshop on Distributed Data Management and Computer Networks, pp.217-235,1977.

[WWH00] Yijie Wang, Yongjun Wang and Shouren Hu, "Parallel execution of multi-join query", Jisuanji Xuebao Vol.23,No.2, pp.177-183,2000.

[YC84] C. T. Yu and C. C. Chang, "Distributed query processing", ACM Computing Surveys,16(4), pp.399-433,1984.

[YGC87] C. T. Yu, K. Guh and A. L.P. Chen, "An integrated algorithm for distributed query processing", Proc. IFIP Conference on distributed Processing, 1987.

[YKB99] Haiwei Ye, Brigitte Kerherve and Gregor V. Bochmann, "Qos-Aware Distributed Query Processing", Database and Expert Systems Applications, 1999. Proceedings. Tenth International Workshop on, pp.923 –927,1999.

[YL90] Clement Yu and Chengwen Liu, "Experiences with distributed query processing", IEEE, pp. 192-199,1990.

[YLG+86] Clement T. Yu, Leszek Lilien, Keh-Chang Guh, Marjorie Templeton, David Brill, and Arbee L. P. Chen, "Adaptive Techniques for Distributed Query Optimization", ICDE, pp.86-93,1986.

[Zha03] Yue (Amber) Zhang, "Variation of Bloom Filters Applied in Distributed Query Optimization", Master's Thesis, University of Windsor, 2003

Vita Auctoris

Name: **Yongmei Zhu**
Place of Birth: Benxi, Liaoning Province, P.R.China
Date of Birth: October 13, 1965

Education:

M.Sc. Computer Science
University of Windsor
Windsor, Ontario, Canada
Sep. 2001 — Jan. 2004

M. Eng., Computer Science
Northern Jiaotong University
Beijing, P.R. China
Sep. 1987 — Mar. 1990

B.Sc., Computer Science
Northern Jiaotong University
Beijing, P.R. China
Sep. 1983 — July 1987

Working Experience:

Graduate/Teaching Assistant
University of Windsor
2001 — 2003

Instructor/Lecturer
Northern Jiaotong University
1990 — 2000

Software Engineer
Bowne Global Solutions, Beijing, PRC
1998 — 2000

Technical Support, Engineer
Beijing Xioatong Electronic Co.
1996 — 1998