2003

Variation of bloom filters applied in distributed query optimization.

Yue (Amber). Zhang
University of Windsor

Variation of Bloom Filters Applied in Distributed Query Optimization

by

Yue (Amber) Zhang

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

Windsor, Ontario, Canada

2003

Your file Votre référence
ISBN: 0-612-84557-5
Our file Notre référence
ISBN: 0-612-84557-5

989071

Abstract

Distributed query processing is important for Distributed Database Systems. Through the past years, the research focus in distributed query processing has been on how to realize join operations with different operators such as semi-join and Bloom Filter. Experiments show that using bloom filters, the hash-semijoin, almost always does better than semi-join for the query processing. However as long as you use bloom filter, you cannot avoid collisions. So in order to get the cheaper processing, some of the past work uses two or more bloom filters to do the hash-semijoin. However several factors still affect the cost and optimization result.

1. How to decide the perfect number of the bloom filters, and what kind of bloom filter should be chosen.
2. There is no way to avoid collisions when utilizing bloom filters.
3. With bloom filter, we cannot keep the exact location information of the joining attributes (loss of join information).
4. With bloom filter, we never can combine the useful composite semi-join in the process.

Taking the idea of PERF join into account, why not use the bloom filter (hash-semijoin) concept but come up with a new kind of filter "Complete Reducing Filter" (CRF), which can avoid the disadvantages of bloom filter, as well as inherit the advantages of it?

We propose and implement a new algorithm called Complete Reducing Filter (CRF) based on PERF join, which can keep the join location information, as well as lower transmission cost (because it's still using the filter concept). At the same time, CRF can combine the composite semi-join into the process, which overcome the impossibility if only using a bloom filter. With the variation of the bloom filter, we try to achieve better performance with lower cost.

Keywords: Distributed query processing, semi-join, bloom filter, hash-semijoin, composite semi-join

To my family

Acknowledgement

I am very happy to take this opportunity to thank people who helped me a lot during the whole process of my graduate study, and towards the achievement of my thesis. It's the only section that I can express my appreciation and love in the thesis.

First, I would like to convey my deep affection to my supervisor, Dr. Morrissey. The work could not be accomplished without her extensive guidance. Besides that, she is kind, nice and supportive, who can be regarded as a sincere friend in study and life. I always feel I am so lucky to be given a chance to know my supervisor and work with her...

Secondly, I would like to thank all my friends and colleagues, who shared with me their precious time to discuss academic issues, provide useful suggestions, and worthwhile experience during the research.

Thirdly, I would like to give my special thanks to my aunt, uncle and cousin, who give me continuous support and guidance through my university years.

Next, I would like to thank my committee members: Dr. Akshai Aggarwal, Dr. Scott Goodwin and Dr. Myron Hlynka, for their time and instructive suggestions and comments.

Thanks, my dear mom and dad, for your valuable care and endless love...

Of course, thanks must be given to my friends and acquaintances who enlighten me and bring cheerful days for me...

Table of Contents

List of Figures

Chapter 1 — Introduction

Database Systems are the corner stone of our current highly developed society. In recent decades, we have achieved great progress in this field, from the first generation — file system to current relational, object-relational or multimedia database systems [Gra96][Ozs00]. With the change of organizations' structure and business style, distributed database systems (such as Mariposa [Ols94], System R* [MOH84], Ingres [Ull89]) are naturally playing an important role for enterprises. Whether catering to the needs of large data warehousing or data mining, or even mobile database systems, effective query processing is a key factor affecting the system's performance. In a distributed database system [OV91][Ols94][Pik97], query processing plays the role. Different algorithms that improve the performance of queries have been developed during the past two decades. The overhead of distributed query processing contains two parts: Local cost and communication cost. Various techniques provide many algorithms to optimize these two aspects for better performance. Most of them take communication cost as the focus for distributed query optimization. Operations such as join, semi-join, hash-semi join and so on are used in various algorithms to achieve optimal results. It has been proved that the **optimal** query processing problem is **NP-hard** [HR94][GZ98][PV88]. So through different search strategies, whether heuristic or dynamic programming, we only try to find a near-optimal strategy for query optimization. The query is usually processed as follows [AHY83][BGW+81]:

1. Initial local processing: All local operations are processed (Selections and projections).
2. Specific operations (Such as semi-join reduction).
3. Final Reassembling (send all needed relations to the final site for final joins).

To consider effectively reducing the communication cost, most of the algorithms focus on the second step. With different operators (semi-join, hash-semijoin or PERF[1] join [LR95]), and different searching strategies (dynamic programming, greedy, or randomized methods), the size of needed relations (or fragments) is reduced, so that cost of transmission to the final site is reduced as much as possible. Different cost models have different criteria, among which response time and total time are often used. Response time cost models consider that each operation is processed in parallel, so the maximum time from sending the query to getting the result is the cost of the processing. The total time model considers the whole time consumed during processing.

The rest of this thesis will be divided into several parts. In chapter 2, the literature review and various strategies for distributed query processing are described. In chapter 3, we introduce the motivation for the thesis, and the methods we propose to use for implementation and evaluation. The experiments and evaluation results will be given in chapter 4. Finally we will give conclusions and look forward to future work.

[1] PERF is Positionally Encoded Record Filter, which was described in details in the paper [LR95].

Chapter 2 — Literature Review (Background)

2.1 Distributed Database System (DDBS)

The use of a database system is motivated by the desire to integrate the operational data and to provide autonomic control over the data. A database system is an evolution from the traditional management systems such as a file system. Database Management systems provide integrated manipulations on data instead of depending on the various applications for random control of the data. With the widespread use of networks, a new kind of database system is coming into being, that is, the distributed database system. It is a combination of a database system with the computer networks [CY91]. As Figure 1.1 shows, for any organization, in order to implement and maintain the necessary source management and data flow, it may need several databases for the whole organization, which are located in more than one place, or even distributed far away from each other. But the most important fact is, they are connected by a network, and communicate with each other and provide correct functions for any upper queries. However, for the users sitting in front of any terminal, who try to find a query result, they even don't notice the lower layer's distributed features. In front of them, it seems a big central database below in the system that provides everything they want.

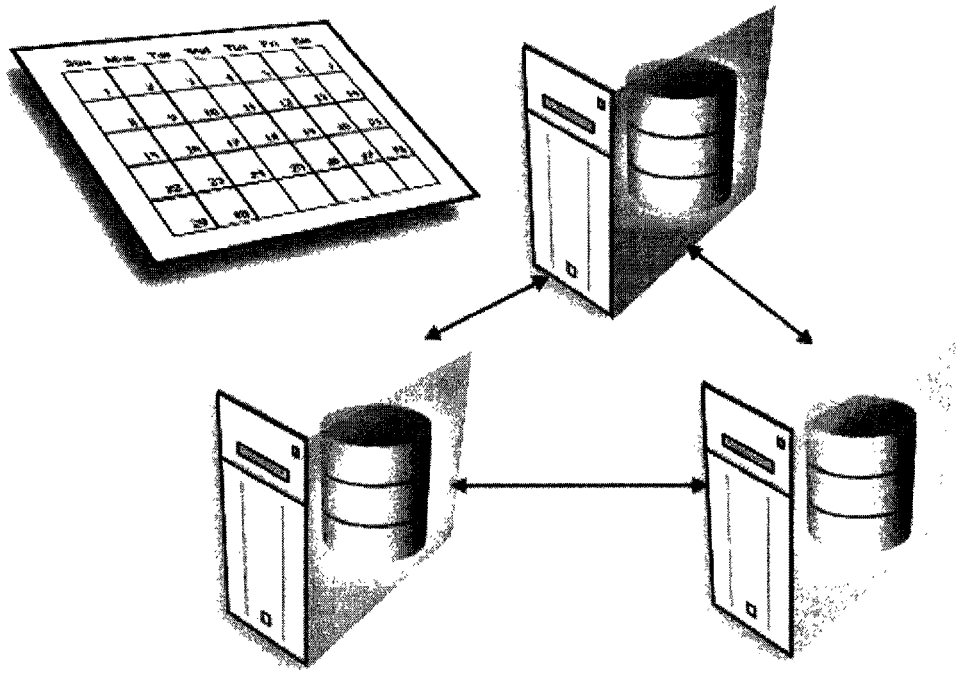

Figure 1.1 Distributed Database Systems

Generally speaking, control of a database is changing from the centralized to the distributed, which brings many advantages together with many issues. In the environment of distributed database systems, the manipulation of data, or data processing can be realized in parallel, with the support of multiple computers geographically distributed. What is a real distributed database system (DDBS)? We can define it as a collection of multiple, logically interrelated databases distributed over a computer network. A distributed database management system (DDBMS) is then defined as the software system that permits the management of the DDBS and makes the distribution transparent to the users. We may get a rough idea about a distributed environment from Figure 1.2. The first upper layer is terminal layer, which can be regarded as clients, who send out queries and fetch information from the lower layers to meet their needs. The second layer can be regarded as servers, which execute commands coming from clients, and send the

outcome back to clients as well. This layer is the intermediate layer, which connects the clients and the data source. Under it, is the data source layer. More than one database is sitting there. If we look at the layout horizontally, we can see different servers have connections to each other. So do the database systems. This horizontal connection between different database systems will fulfill the functions of distributed database systems, such as data sharing, replication and so on.

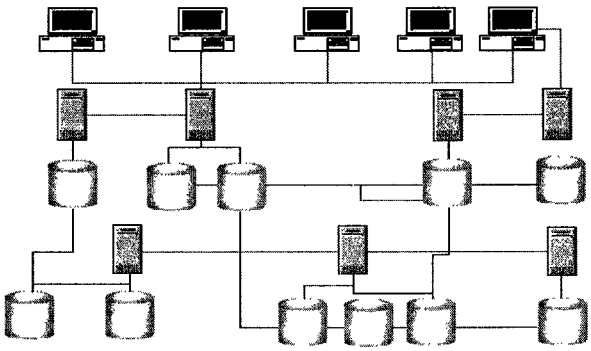

Figure 1.2 Layout of Distributed Environment

Basically, distributed database systems can have such attributes as the following:

The database is stored on several computers, or sites, which are connected via a network. Different sites have different data, functions and relations, and each one has to maintain a local database, but at any time, it can participate in a global query. With such systems, data sharing between sites becomes easy, and in many systems, local control in function over local data is maintained to allow autonomy. With replication, reliability and availability can be ensured. At the same time, such systems have increased complexity, more development costs, greater potential for coding errors, and increased overheads due

to communication via a network. Also we have to think much more about security over the network.

Among complex issues, the most important is query processing, which is the core of a database system for effective response. So what is the optimal execution plan for a query in a distributed database system? How do we get the best outcome in a distributed environment? Query optimization is what we have to focus on.

2.2 Query Optimization Processing

As mentioned above, in distributed database systems, a large number of parameters affect the performance of distributed queries [Kos00][Pik97]. In particular, the relations involved in a distributed query may be fragmented [Seg86] and/or replicated to reduce the communication overhead costs. Furthermore, with many sites to access, query response time can be very high. We just draw our attention to relational distributed database systems, which are popular.

The main function of a relational query processor is to transform a high level query (direct user input query) into an equivalent lower level query. Then the lower level query processing implements the execution plan. The transformation from the high level query to the low level query should be correct and efficient. Because a relational query may have multiple equivalents and correct transformations into relational algebra, where each execution strategy can lead to a great difference in performance and cost, the main problem of query optimization is to find or select the strategy that minimizes the cost. So

one of the core components for a query optimizer is its search strategy or enumeration algorithm. Among the operations on a database, select-project-join is the most commonly used. In the literature, many different methods are proposed by different researchers on such operations, especially on joins. The basic steps for distributed query processing are [LPP91]:

- ✓ Sequencing optimization: The best sequence of binary joins is selected to execute n-ary joins
- ✓ Materialization: Each relation has to be retrieved. It may exist in multiple copies in the system or may be fragmented horizontally or vertically. The optimal site for materializing is chosen.
- ✓ Distribution: The optimal allocation of binary join executions as well as the storing of intermediate result relations among the available sites is determined.
- ✓ Execution strategy: A binary join can be implemented by means of a number of techniques.

In order to achieve optimal or near-optimal execution plans, different kinds of operators are introduced to deal with join operations, such as semi-join, bloom filter [AHY83][BGW+81][TC92][WLC91]. Most research assumes that the transmission cost via a network is the most expensive part of query processing cost except the local cost for local processing. There are different situations based on different network topologies. In some rapid networking environments, local cost is comparable with transmission cost. Whatever the case is, the join operation is one of the most time-consuming operations, so

current research pays lots of attention to how to reduce the cost for join operations in distributed database system, such as what the join order is, what kind of operators are involved and so on. In the following sections, we give more details of powerful algorithms occurring in query processing.

2.3 Algorithms

Different algorithms based on different views can be classified into the following categories: join, semi-join, hash-semijoin, PERF join and interleaving join with semi-join.

Join Operator

Join is a very time consuming operation in database systems [Ull89]. When we migrate from centralized database systems to distributed database systems, the join operation becomes the most costly process. Look at Figure 2.1. Suppose we have two relations that are residing separately on different sites. In relation R, we have the information about ID and Name, while in relation S we have ID and Age. In order to get full information on ID, Name and Age, we need to join these two relations in the distributed database system. Based on the joining attribute: ID, we do the join operation, and get the final relation as shown in this figure. As we can see from this example, even though the final result of the join operation is always same, the way to implement the join operation varies a lot. Because relation R and relation S are not residing in the same database, we can have many means to do that. For example, we can first send the whole relation R to the site where relation S is residing, then do the join operation locally at the site of S, and finally send the join result to the query site. So many authors focus on how to minimize the cost

of join operations. The main focus of the join method in System R* is as follows [MOH84]:

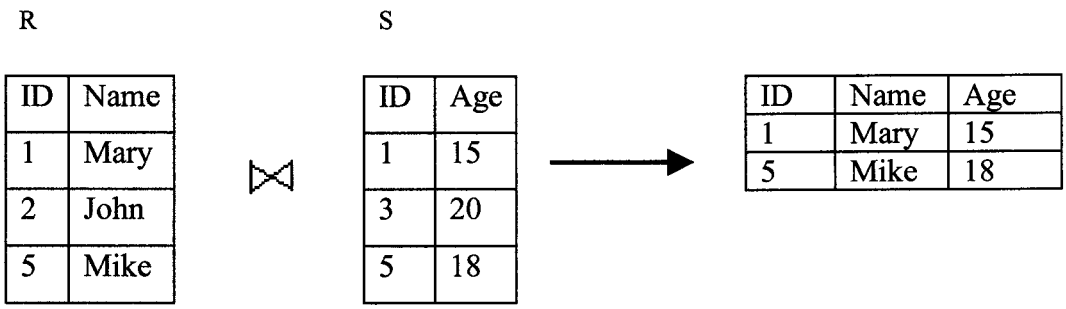

Figure 2.1 Join

Join order — Same as in the centralized case, the optimizer has to select the join ordering.

Access method — When access each fragment, do we use clustered index or sequential scan?

Join method — Use merge join or nested loop join?

The decisions for these three issues are based on statistics and formulas used to estimate the size of intermediate results and access path information. In the meantime, the optimizer has to select the join site and the method of transferring data between sites.

Join site — Decide the best site or required site for the join operations. There are three candidate sites for the join operation: the site of the first relation, the site of the second relation and a third site, such as the query site where the join result needs to be presented.

Inner table transfer strategy — When we need one relation (called inner table or slave relation) to join with the other one (called outer table or master relation), what is the inner table transfer strategy. Just transferred as needed (called Fetch-as-needed) or wholly transferred (called Ship-whole)? Ship-whole generates a larger data transfer but fewer messages than fetch-as-needed. It is obviously better to ship whole relations when they are small. However, when the relation is large and the join has good selectivity (only small parts of tuples matched), we should use fetch-as-needed.

The R* algorithm uses the total time as the standard for the strategy. It can be regarded as an exhaustive search among all alternatives that are defined by the permutation of the relation join order, join methods, result site, access method and intermediate site transfer mode.

Semi-join

The semi-join has become the most popular operator recently to replace joins, because it effectively reduces the inter-site's relation size and minimizes the cost of transmission. What is a semi-join? Basically, a semi-join has the following steps to go through [AHY83][BGW+81] [Gra96] [HR94] [WLC91]:

1. Local processing (Selection-Projection).
2. On the joining attribute, give the projection of that attribute of one relation.
3. Transfer the projection to the other relation and execute join between the projection and the second relation.
4. Finally transfer the reduced relation to the final site for joining.

Suppose we wish to join two relations R_i and R_k, which means we want to execute $R_i \infty R_k$ over the join attribute j. A semi-join from relation R_i to relation R_k over the join attribute j, is executed as follows:

1. Project relation R_i over attribute j to get the projection d_{ij}.
2. Ship the projection d_{ij} to the site of relation R_k.
3. Execute $d_{ij} \infty R_k$.

The semi-join reduces the size of R_k by eliminating the tuples that cannot be part of $R_i \infty R_k$. The reduced R_k can now be shipped to the query site with savings on transmission costs. In a similar way we can reduce R_i using the reduced R_k. Here is an example. Two relations R_i and R_k on joining attribute j:

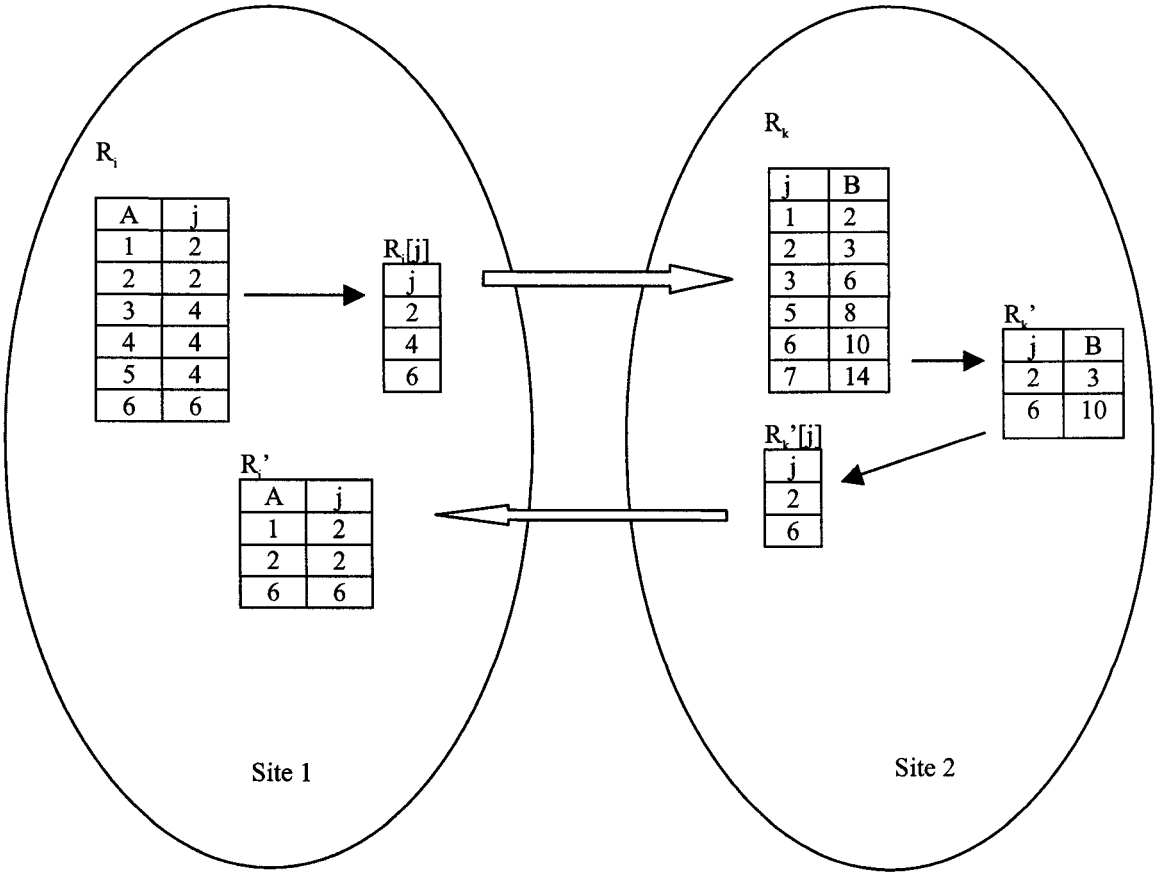

Figure 2.2 Semi-join

First, we project relation R_i on the joining attribute j, we then get the projection $R_i[j]$ with 3 tuples valued as 2, 4 and 6. Then we ship the projection $R_i[j]$ to site 2 and join it with relation R_k. Therefore, we get reduced relation R_k as R_k'. So far we finish the first semi-join from R_i to R_k. Similarly, we project R_k' on the joining attribute j. Ship the projection $R_k'[j]$ back to site 1. At site 1, we join projection $R_k'[j]$ with R_i, so that we implement the semi-join in the other direction, and reduce R_i as well.

SDD-1 [BGW+81]

SDD-1 was the first method in Distributed query processing to use the semi-join as the reducer to minimize the cost. It uses the set model as the estimation of cost and benefit. In the SDD-1 distributed system, queries are submitted to SDD-1 in a high level procedural language called *Datalanguage*. Optimization begins by translating each Datalanguage into a relational calculus form called an *Envelope*. Then Envelopes are processed in two phases.

Phase 1: Subset (Database) — Execute relational operations at various sites of the distributed database in order to delimit a subset of the database that contains all data relevant to the envelope. The semi-join operator is used in the reduction phase to reduce the size of the relations that do not satisfy the qualification of the query.

Phase 2: Transmit the reduced relations to one designated site.

AHY [AHY83]

In this paper, instead of computing the joins immediately, we first reduce the sizes of each relation by possible restrictions and projections. If one relation has the join

attributes, we use an operation called semi-join to delete the unnecessary tuples. To compute a semi-join, the unique values of the joining attribute of one relation are sent to the other relation. It is cheaper to compute this semi-join than the complete join. In the result node, the complete join will be computed after the reduced relations have arrived by concatenating matching tuples on the joining attributes. The data transmissions are used for reducing a relation and the transmission of the reduced relation to the query computer form a schedule for the relation. We first give some notation. d_{ij} represents the projection of relation i on joining attribute j. For example, as shown in Figure 2.2.1, d_{21} means projection of relation 2 on joining attribute 1. Attributed d_{21} is sent to attributed d_{31}. A semi-join is performed on relation R_3. The reduced d_{31} attributed is sent to relation R_1 in parallel with attributed d_{22}. Further relational operations reduce the size of relation R_1. Finally the reduced relation R_1 is sent to the result node.

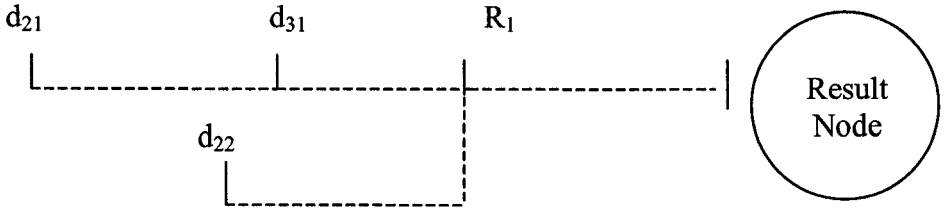

Figure 2.2.1 AHY

Hevner and Yao introduced and investigated algorithms Parallel and Serial, which respectively compute minimum response and total time schedules for simple queries.

A general query has more than one joining attribute. So, a relation can be reduced in size by semi-joins on different joining attributes. However each time it considers just

a simple join, which means all the relations are involved in the semi-join with a single attribute. This may not give the optimal solution from the algorithm.

Hash-semijoin

The idea of hash-semijoin is to use a new operator hash-semijoin to replace the semi-join in order to reduce the transmission cost of semi-join, so as to get the reduced result more efficiently [CCY92][Gra96].

Step 1: Initialize a bit array of F bits all to be 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

Step 3: Transmit the bit array to the site of R_i.

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.

For Example: Hash (B attribute value) → address

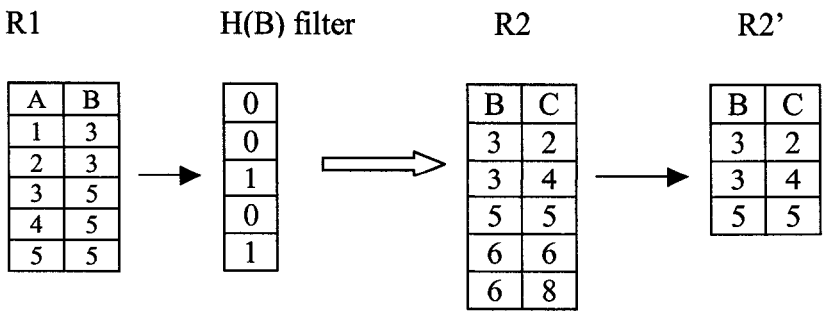

Figure 2.3 Hash-semijoin

For each tuple in relation R_1, we use the hash function H(B) $\rightarrow$ address. Because we only have value 3 and 5 in the column B, so in the filter bit array, at the addresses 3 and 5, we set '1', the other bit we set '0', which means we do not have any value that can be hashed to this address. Then we send the H(B) filter to relation R_2. Based on the same hash function, we probe relation R_2 row by row. If the correspondent address in the H(B) filter is '1', we keep this tuple in R_2, otherwise, we get rid of the tuple, because it can not be in the final join result anyway. Therefore, we get the reduced relation R_2 as R_2', as shown in Figure 2.3.

The hash-semijoin is proposed for more cost-effective distributed query processing. The search filter in the hash-semijoin achieves considerable savings in the cost of a semi-join operation. However it only works on tree queries, and the performance is tightly related with the hash functions (bloom filters). For the bloom filter, the main advantages are cost reduction and less storage demand; Comparably, Value vector [Mul96]: $V_A = (n_1,\ldots,n_m)$ is defined such that n_i is the number of tuples in a relation which have value vi for attribute A. To join two relations, which have not been reduced, the value vector method will be very accurate. For already reduced relations, the bloom filter method does not rely on assumptions about independence of attributes. So bloom filter method is more reliable. In the value vector approach, one must store the number of tuples, which contain each possible value of each attribute. In the bloom filter approach, one scan of the relation is enough to build the filter (for all attributes desired). Only a few thousand bits is needed for storage [Mul96].

Two-way Semi-join

Two-way semi-join is an extended version of the semi-join for more cost-effective distributed query processing [Seg86]. If we denote the two-way semi-join of Ri and Rj on attribute A as $R_i \leftarrow A \rightarrow R_j$, then $R_i \leftarrow A \rightarrow R_j = \{R_i — A \rightarrow R_j, R_j — A \rightarrow R_i\}$ It is computed by the following steps:

1. Send $R_i[A]$ from site i to j.
2. Reduce R_j by eliminating tuples whose attribute A does not matching any of $R_i[A]$. During reduction of R_j, partition $R_i[A]$ into $R_i[A]m$ and $R_i[A]nm$ where $R_i[A]m$ is the set of values in $R_i[A]$ which match one of $R_j[A]$ and $R_i[A]nm$ is $R_i[A]$- $R_i[A]m$.
3. Send either $R_i[A]m$ or $R_i[A]nm$, whichever is less in size from site j back to i.
4. Reduce R_i using either $R_i[A]m$ or $R_i[A]nm$. If $R_i[A]m$ is used, then tuples whose attribute A does not matching any of $R_i[A]m$ are eliminated. If $R_i[A]nm$ is used, then tuples whose attribute A matches one of $R_i[A]nm$ are eliminated.

Global Semi-joins

In [RK91], a sequence of forward semi-joins with a sequence of backward semi-joins is applied to propagate the benefit of semi-joins on relations. In the process, a structure known as a connector is used, which records the former semi-join's effect. This is a small table, which can easily fit in the memory for the next step semi-join. However, this method is a serial method, which excludes the possibility of processing in parallel, and it omits the local processing cost, which may be high. Comparably, in [WLC91][CCY92], a parallel semi-join execution method is proposed. The goal of one-shot semi-join execution is to remedy the inefficiency of traditional semi-join processing algorithms, which favor sequential execution of semi-joins. Under this new method, the initial local

processing and final join processing for a distributed query remain the same. But, the query optimizer has to decide on a set of semi-joins to be executed first and the semi-join processing step is further partitioned into three phases, namely, the projection phase, the transmission phase and the reduction phase. In the projection phase, a relation R_i is scanned once to generate all the necessary semi-join projections. Then all the semi-join projections are transmitted in parallel to the corresponding sites to perform semi-joins in the transmission phase. After the transmission, for each semi-join to a relation R_i, its semi-join projection is available at the site where R_i resides. Therefore, global semi-join optimization is possible. Global semi-join optimization is impossible if semi-joins are executed sequentially.

Domain specific Semi-join

Many query optimization algorithms proposed for fragmented databases apply semi-joins to reduce the size of the fragments of joining relations before they are sent to a final processing site. When semi-joins are employed in such a system, they have to be performed in a relation to relation or a relation to fragment manner to avoid eliminating contributive tuples. Many algorithms [AHY83][BGW+81][CL84] only propose semi-joins for non-fragmented databases. In order to improve the semi-join associated with the joining fragmented relations, domain specific semi-join is introduced [CL90]. It may be performed in a fragment to fragment manner and provide more flexibility in distributed query processing. In horizontally partitioned databases, a domain is a set of values. Tuples of a relation are horizontally partitioned into disjoint subsets called horizontal fragments. A fragmented relation R_i can be reconstructed by performing a union operation on all its fragments. $R_i = \cup_k R_{ik}$. So a join between two fragmented relations R_i

and R_j is equivalent to a union over joins between each fragment of R_i and each fragment of R_j: $(\bigcup_k R_{ik})\,[A=B]\,(\bigcup_m R_{jm}) = \bigcup_{k,m}(R_{ik}[A = b]R_{jm})$

When using the traditional semi-join in fragmented relations this way, it is easy to lose contributive tuples for the fragment join. The domain specific semi-join operation $R_{ik}(A=B]R_{jm}$, A, B are the joining attributes and R_{ik} R_{jm} are two fragments of the joining relations R_i and R_j, is defined as

$R_{ik}(A=B]R_{jm} = \{r | r \in R_{ik}, r.A \in R_{jm}[B] \cup (Dom[R_j.B]-Dom[R_{jm}.B])$. Here, $Dom[R_j.B]$ represents the domain of the joining attribute B in relation R_j, while $Dom[R_{jm}.B]$ represents the domain of the fragment R_{jm} on joining attribute B. The whole formula means when we semi-join the two segments of each correspondent relation, the domain specific semi-join result is the semi-join result of the two segments' normal semi-join union the semi-join by the first segment with the remaining part getting from the domain of the joining attribute on second relation minus the domain of the second fragment on the joining attribute.

Then for a fragmented relation R_i involved in multiple joins, the domain specific semi-joins on any fragment of R_i are permutable if the restricting relations are fragmented by their joining attributes. Let R_{ik}' be R_{ik} (A=B) R_{jm}, the fragment R_{ik} after the domain specific semi-join reduction by R_{jm}. We also assume $Dom[R_i.A] = Dom[Rj.B]$, then

$$R_{ik}' = |R_{ik}(A=B]\,R_{jm}| = |R_{ik}|\frac{|(1-|Dom[R_{ik}.A] \cap Dom[R_{jm}.B]|}{|Dom[R_{ik}.A]|} +$$

$$|R_{ik}|\frac{|Dom[R_{ik}.A] \cap Dom[R_{jm}.B]\| R_{jm}[B]|}{|Dom[R_{ik}.A]\| Dom[R_{jm}.B]|} = |R_{ik}|\frac{|R_{jm}[B]|}{|Dom[R_{jm}.B]|}$$

Composite Semi-joins

A composite semi-join is a semi-join in which the projection and transmission involve multiple columns [PC90]. In most of the algorithms multiple semi-joins may be performed with common source and common result sites. In this situation it may be beneficial to do the semi-joins as one composite rather than as multiple single column semi-joins. The paper looks at two classes of algorithms that may be improved with composite semi-joins. The first is the variations on Algorithm General response time version [AHY83], and selectivity is the main estimation method. The second is variations on Algorithm W. It uses "worst case elimination" to estimate attribute sizes after semi-joins. However both are static algorithms, not dynamic. For example, we have two relations 1 and 2 shown in Figure 2.4. Two common joining attributes 1 and 2 exist in both relations, which are denoted as D11, D12 and D21, D22. If we do only common semi-joins on the two joining attributes one by one, we will not reduce relation 2 at all. However, by comparison, if we do the composite semi-join, we'll reduce the redundant tuples greatly.

We come to the following conclusion: Composite semi-joins are not always the best, however by combining semi-joins and composite semi-joins in one algorithm, it may be the best approach.

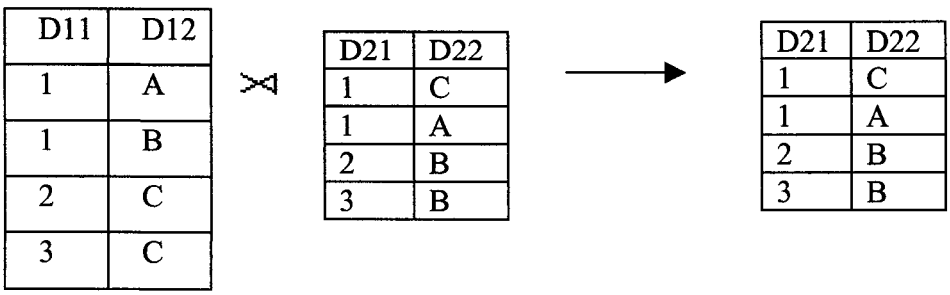

Figure 2.4 Semi-join Only

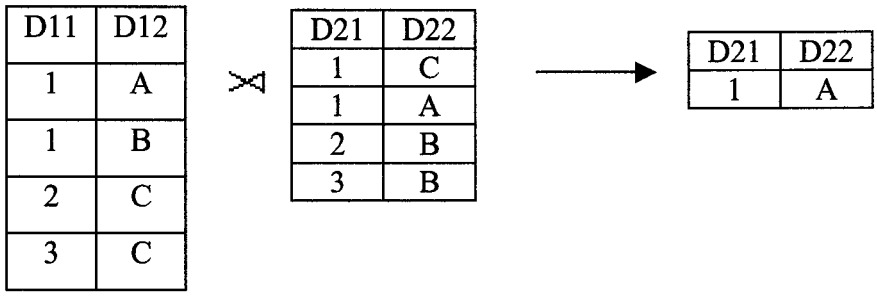

Figure 2.5 Composite Semi-join

PERF join [LR95]

A PERF is based on the relation tuple scan order instead of hashing. Hence it doesn't suffer any loss of join information incurred by hash collisions [GZ98]. The basic idea of PERF join is as follows: in 2-way semi-join, relation S is reduced by a semi-join with projection of relation R, P_R. But instead of transferring $P_{S'}$ 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. Over semi-join or bloom-join, it is possible to apply PERF join among the partially reduced relations to further decrease the amount of spurious data being transmitted to the query assembling site, which in turn lowers the overall query response time.

PERF preserves the complete join information and minimizes network and storage overhead. It leads to cheap local join processing cost, especially when buffer memory is scarce. It also can handle the inequality join query and cyclic join query [LR95].

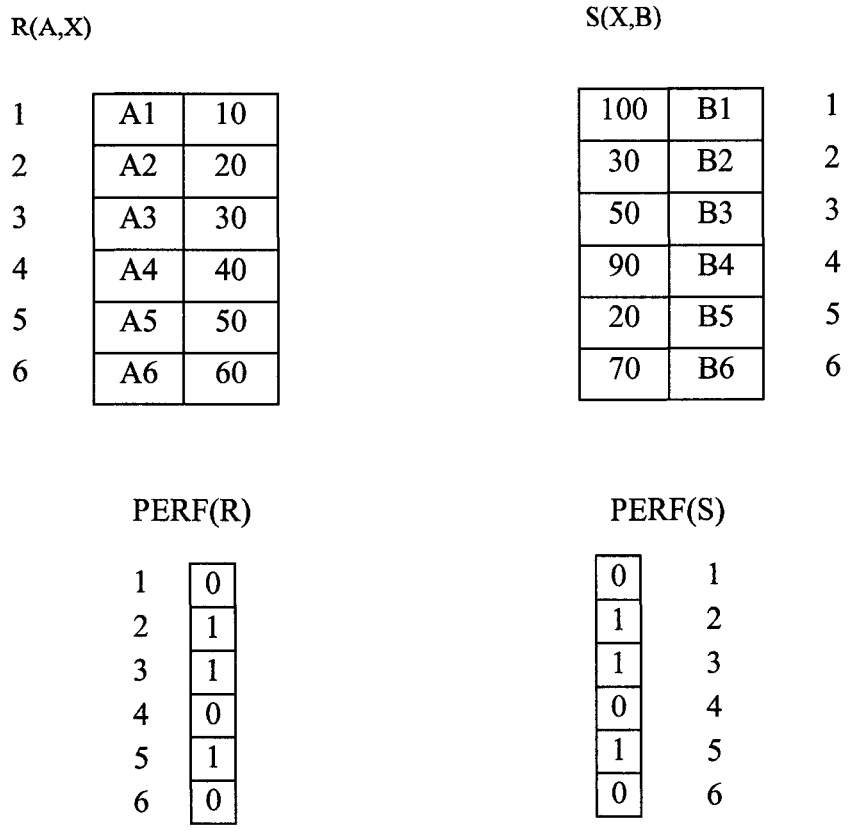

Figure 2.6 PERF Join

From the above figure, we can know clearly how to get the PERF vector of each relation. In the two relations R and S, X is the joining attribute. So we can get projections on joining attribute X of each relation denoted as P_R and P_S. When we begin to produce PERF(R), we pick each tuple in the projection P_R, and look into P_S to see whether its value exists in P_S. If it does exist, we set correspondent '1' in the bit vector, otherwise, we set '0'. After going through all the tuples in the projection P_R, we get PERF (R). Similarly, we scan each tuple in projection P_S, and search that value in projection P_R. If this value exists in the projection P_R, we set '1' for the bit vector, otherwise '0'. Finally, we will get the PERF (S).

Interleaving join with semi-join

In distributed query processing the conventional approach to reduce the amount of data transmission is to first apply a sequence of semi-joins as reducers and then ship the resultant relations to the final site to carry out the join operations. In view of this fact sometimes if we mix semi-join and joins, we can get better performance. In [CY91], an algorithm first establishes the join sequence, and it uses heuristics to insert semi-join operators.

Chapter 3 — Motivation & Evaluation Methods

3.1 Motivation

From experiments in [MO99], we can tell that it is almost always better to use hash-semijoins than semi-joins. New algorithms which use hash-semijoins exclusively should give better performance than simply taking the schedule produced by a semijoin-based algorithm and replacing the semi-joins with hash-semijoins. However, using hash-semijoin or bloom filter can never avoid false drop or so called collisions. In order to minimize the collisions that happen using the bloom filter, in [Lia99] two bloom filters are used, and experiments show that it is almost always better than using only one bloom filter. Will it be better to use more bloom filters? How should we choose the bloom filters? In fact, it so far has not been well formulized (only in [Mul96], some experimental data were given). Generally speaking, as long as you use bloom filters, you cannot avoid collisions. Why not use some new filter that can avoid collisions to reduce relations, so that we can get the benefit of using bloom filters, as well as avoid further collisions to achieve better performance? Taking [CCY92][Lia99][LR95][Ma97][MOL00][WLC91] into consideration, if we can find and use a new kind of operator (filter) which does not have any collisions to produce the schedules, we may not only get the benefit from filter based methods, but also avoid the cost produced by collisions of bloom filters. Another consideration is, whenever you use bloom filter, you cannot use composite semi-join as well. Based on the general idea from PERF join [LR95], I propose the new algorithm called Complete Reducing Filter (CRF). Using CRF algorithm, we can not only get the benefit of filter based methods, but also can combine composite semi-join with reducing filter to get better performance.

Complete Reducing Filter (CRF)

1. Apply one bloom filter: Reduce each remote relation R_i with bloom filters generated from a subset of the other joining relations. The resulting relation R_i' might still contain some non-joining tuples.
2. Send join attributes: Transmit join attributes projection(s) $P_{Ri'}$ in parallel to the assembling site. Combining the composite semi-join, join the $P_{Ri'}$ and generate a CRF(R_i') for each corresponding R_i'.
3. Receive CRF: Ship CRF(R_i') and combine with each R_i'. R_i' will be fully reduced into a relation R_i'' which contains only the matching tuples.
4. Transmit matching tuples: After excluding the join attributes, send each R_i'' in parallel to the assembling site to compose the final join result.

As to the step 1 of CRF algorithm, optionally, we can work directly on the original large-size relations, because the input (relations to be dealt with) is transparent to CRF. We can take original relations as input to test CRF alone, also we can take the result of bloom filter as input to test whether it is also beneficial to apply CRF on the relations already processed.

First, let us look at a simple example about how the CRF works.

Suppose we have two relations R_1 and R_2. In each relation, they have joining attributes J. In order to produce the CRFs as we planned, we go through the following steps:

1. At each local site, project $R_{i\,(i=1\text{ or }2)}$ on the joining attribute. Here the joining attribute is only J.

2. Make up the special form of projections locally with joining position information, as shown in Figure 3. 1.
3. Only transfer the joining attribute column to an assembly site.
4. Scan the projections across, and produce each CRF
5. Send back each corresponding CRF and reduce each relation locally
6. If at the assembly site we have more than one projections from one relation, we use the logical operation AND to combine the filters, so as to implement the composite semi-join operation.

For example, we have CRF filters on relation R_i, totally the numbers of such filters for R_i is equal to j. Therefore we can collectively get the final CRF filter on R_i, $CRF(R_i)$, which is: $CRF(R_i) = \Pi_{k=1 \text{ to } j} CRF(R_i)_k$

Site of R_1

J	Pos
A	1
B	2
C	3

Site of R_2

J	Pos
C	1
A	2
D	3

Figure 3.1 Projection and Position information locally produced

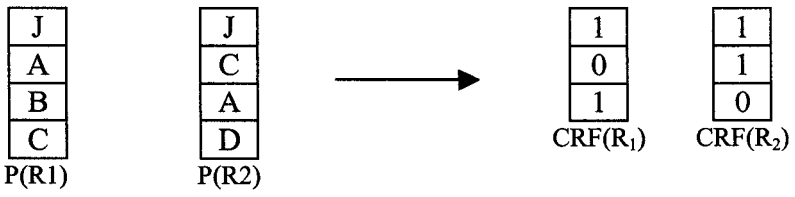

Figure 3.2 Scan projections and produce filters

Here, we recap the whole algorithm with a comprehensive example:

Suppose we have three relations R_1, R_2 and R_3. In relation R_1, we have three attributes A, B and C, while B and C are joining attributes. In relation R_2, we have four attributes B, D, E and F, while B and F are the two joining attributes. In relation R_3, we have two attributes F and G, while F is the joining attributes.

Here are the tables:

R_1

A	B	C
0	3	2
1	5	8
2	7	4
3	9	6

R_2

B	C	D	E	F
3	8	Y	Mr.	X
5	2	N	Ms.	Q
7	3	A	Mr.	P
9	6	Y	Ms.	W

R_3

F	G
X	F
W	M
Y	M
W	M
Z	F

Figure 3.3 Tables of Relations

1. First we project each relation on the joining attributes to get every joining attributes' projections, $P_{R1(B)}$, $P_{R1(C)}$, $P_{R2(B)}$, $P_{R2(C)}$, $P_{R2(F)}$ and $P_{R3(F)}$.
2. Choose the relation with maximum degree to begin. In the example, because relation R_2 has three joining attributes, so it has the maximum degree. We start the algorithm from R_2.

   On each joining attribute of R_2, we scan the other projections of the same joining attributes and produce the CRF of R_2 on each joining attribute. So we get $CRF_{R2(C)}$, $CRF_{R2(B)}$, $CRF_{R2(F)}$ and the composite CRF filter $CRF_{R2(B,C)}$. When we finish the

individual CRF filters of R2, we do the AND operation, to produce the $CRF_{R2,}$ as shown in Figure 3.3, the left frame.

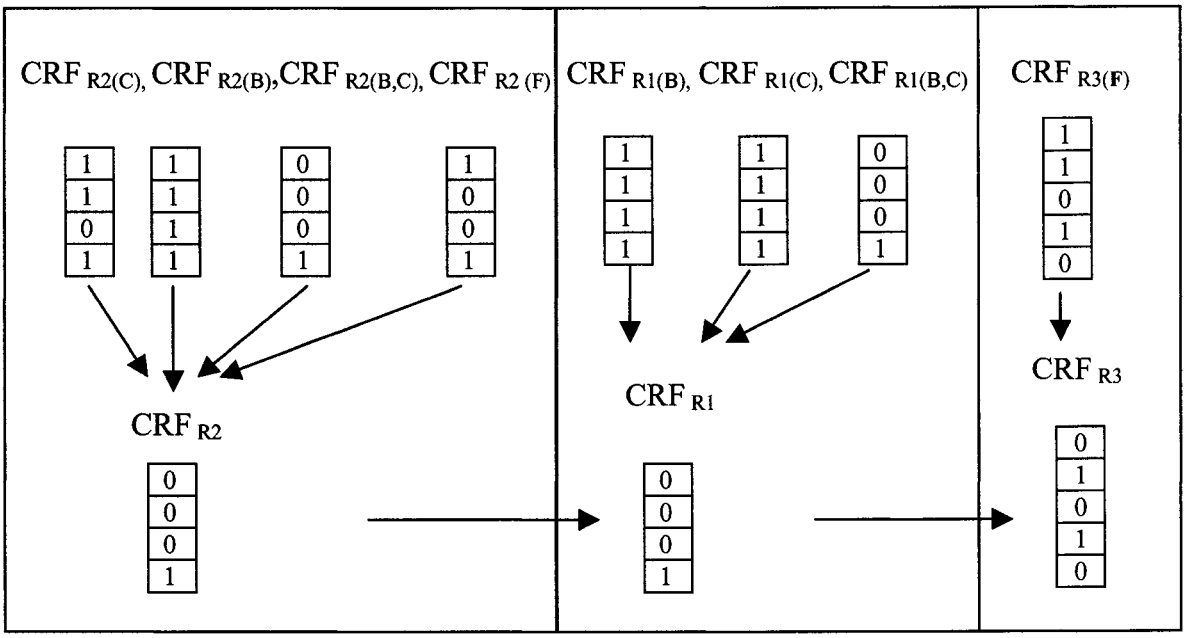

Figure 3.4 Final filters for each relation

Similarly, based on the related projections of other relations and available $CRF_{R2,}$ we get $CRF_{R1.}$ Here the existing of new produced CRF_{R2} will provide the updated information about R_2's projections needed for the following steps. Next based on the previous CRF_{Ri} and correspondent projections, we get $CRF_{R3.}$

3. Send the CRF_{Ri} back to each site.
4. Locally reduce each relation.

After the whole procedure, we are done with the reduction, and finally we can send the totally reduced relations to the assembly site for real join.

3.2 Evaluation Methods

In order to measure the performance of our algorithm, we need to find the objective comparable algorithms to carry on the evaluation.

1. Is it beneficial to apply this algorithm alone on relations involved?
2. It is a filter-based algorithm. Does it really outperform the popular bloom-filter algorithm? If no, what's the condition to apply this algorithm for better performance?

Due to the two main concerns, we choose the following two comparison methods:

a) Compare the experimental result without applying CRF (e.g. Initial Feasible Solution) with that using CRF

b) Compare the experimental result using bloom filters with that using CRF

There are two ways to implement the CRF algorithm:

1. Sequential method
2. Parallel method or also called One-shot method.

Sequential method: Send the projection from one site to other, to get CRF(R_i) in order.

Parallel method: Send all projections of each relations to a third site, to get CRF(R_i) at one-shot. In my work, I want to find out how to implement the algorithm so that it is more cost-effective.

Concerns:

1. The local cost to implement the CRF algorithm (Sort-Scan).
2. In run time, the different cost from the sequential method and from parallel method.

Chapter 4 — Experiments & Evaluation

In this chapter, we present our experimental scenario and summarize the experimental results under such an environment.

4.1 Experimental System

The experimental System includes 3 parts: Database related statistical information generator (create_query.exe), true relations generator (relbuilder.exe), and the proposed algorithm producer (test.exe). The statistical information generator (create_query.exe) and true relations generator (rebuilder.exe) were created by previous colleagues in Database Group of Windsor University, and revised by me. The CRF algorithm producer (test.exe) is fully implemented by me.

Methodology

The framework of evaluating this Algorithm (CRF) is based on objectives:

1. To measure the performance enhancement of this algorithm in terms of response time
2. To measure the performance under a wide variety type of queries

The whole process looks like this way:

User query → create queries and statistical information for the whole database → create relations which meet such requirements → Execute Algorithm CRF → measure the performance

An example of query statistical table is shown below:

User query type: 6 relations and 2 joining attributes

6	2

Database Statistics:

$S(R_i)$	$S(d_{i1})$	$\rho(d_{i1})$	$S(d_{i2})$	$\rho(d_{i2})$
2000	0	0.000000	867	0.704878
2000	783	0.760194	0	0.000000
3200	683	0.663107	0	0.000000
1000	918	0.891262	0	0.000000
900	730	0.708738	895	0.727642
1900	0	0.000000	986	0.801626

Figure 4.1 Database Statistical Information

The size of relation R_i is denoted as $S(R_i)$. The size and selectivity of each individual attribute are represented by $S(d_{ij})$ and $\rho(d_{ij})$ respectively. Here, we recap the definitions on relation size, attribute size and selectivity.

$S(R_i)$ is the size of relation R_i. Suppose cardinality of R_i is $|R_i|$, width of a tuple in R_i (in bits) is $W(R_i)$, $S(R_i) = |R_i| * W(R_i)$.

$S(d_{ij})$ is the size of joining attribute j of relation R_i. d_{ij} represents the joining attribute, the cardinality of d_{ij} is $|d_{ij}|$ (or $|R_i|$, when we don't eliminate the duplicates), then the width of the join attribute in bits is $W(d_{ij})$, $S(d_{ij}) = |d_{ij}| * W(d_{ij})$.

$\rho(d_{ij})$ 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 $Dom(d_{ij})$, $\rho(d_{ij}) = \frac{|d_{ij}|}{Dom(d_{ij})}$. The selectivity is regarded as high when $\rho(d_{ij})$ is small.

For example as shown in Figure 4.1, the size of R1 is 2000, which doesn't have joining attribute 1, so the size of the projection on joining attribute 1 and the selectivity are both 0. However it has joining attribute 2, the size of the projection of relation R1 on joining attribute 2 is 867 and the selectivity is 0.704878.

Test-bed

In order to measure the performance of utilization of the CRF algorithm, we investigate the following characteristics in this thesis:

1. The number of relations involved in the query.
2. The number of possible joining attributes involved in the query.
3. The selectivity of the attributes in the query.
4. The domain size.
5. The number of tuples in a relation.

However what we focus on and try to answer in this thesis is:

1. How does the number of relations in the query affect the performance?
2. How does the number of joining attributes in the query affect the performance?
3. How does the selectivity of the attributes affect the performance?
4. How does the relation size affect the performance?

As mentioned in the last motivation section, the real implementation of such an algorithm can be done with a parallel method and a sequential method. However in this simulation, past research suggests that the parallel method should have better performance than the

sequential method. Because the operations are in parallel, the response time is less, and the parallelism is maximum, which is quite suitable for any real distributed database system. So in all my following experiments, I am using the parallel method to simulate the whole process and get the experimental results.

4.2 Experimental Result and Evaluation

Because in the real 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. The selectivity is classified into 3 categories: Low (0.7 – 0.9), Medium (0.4 – 0.7) and High (0.1 - 0.4). Each relation in the query consists of 500 to 6000 tuples, while the attribute domain contains 500 to 1500 distinct values. The experiments carried out are classified into three parts based on the selectivity of all joining attributes in the test queries. Fifty queries were constructed and executed using the algorithm CRF, or in other words, each type of query will be run for 50 times (runs).

Results

4.2.1 Effect of the number of relations

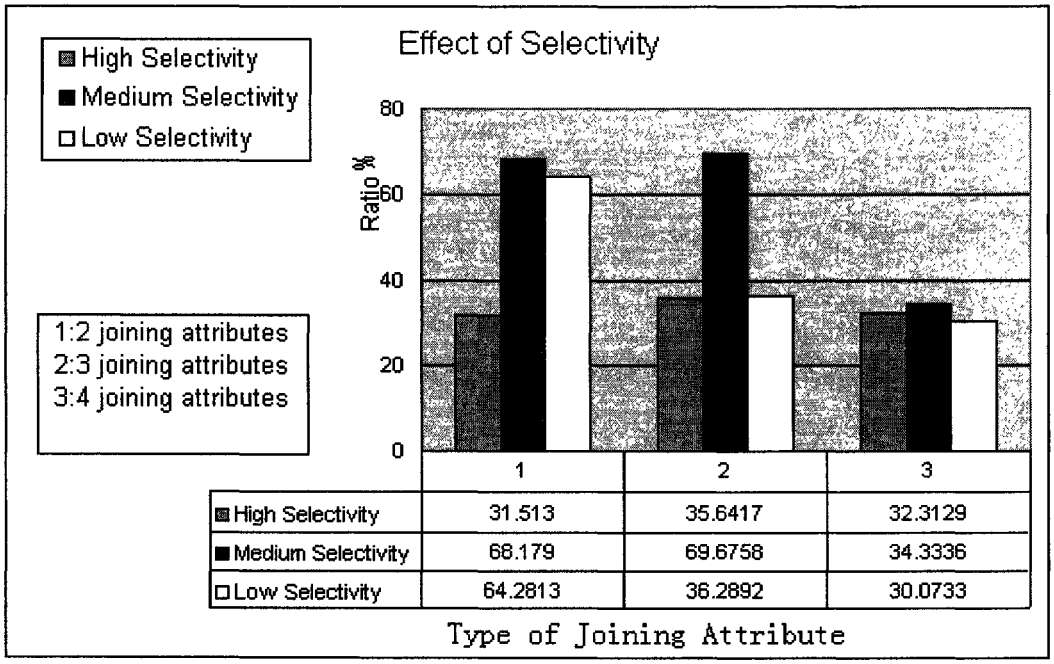

Figure 4.2.1.1 Three Relations

As Figure 4.2.1.1 shows, when we deal with three relations in the simulation, medium selectivity always produces a higher beneficial rate than other selectivity, which is represented in purple by sequence. Generally, when the numbers of the joining attributes increase from 2 to 4, the beneficial rate shows deterioration for low selectivity. But for high selectivity or medium, three joining attributes case gives the relatively best performance. And one more feature of the 3 relations case is that the beneficial rate changes greatly. It can vary from about 30% to about 70%.

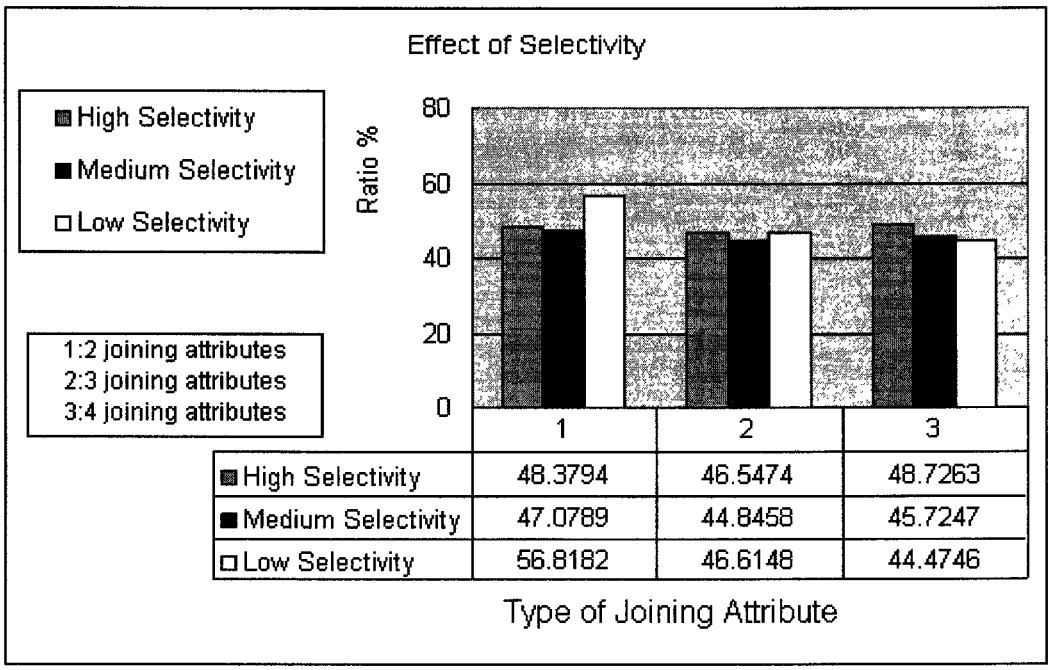

Figure 4.2.1.2 Four Relations

When we increase the relation numbers to 4, performance of the simulation changes a little bit. With the increase of the numbers of joining attribute, the beneficial rate doesn't get worse greatly, under medium or high selectivity, it may be better than or relatively good. But compared to 3 relations case, the beneficial rate for any selectivity doesn't vary a lot.

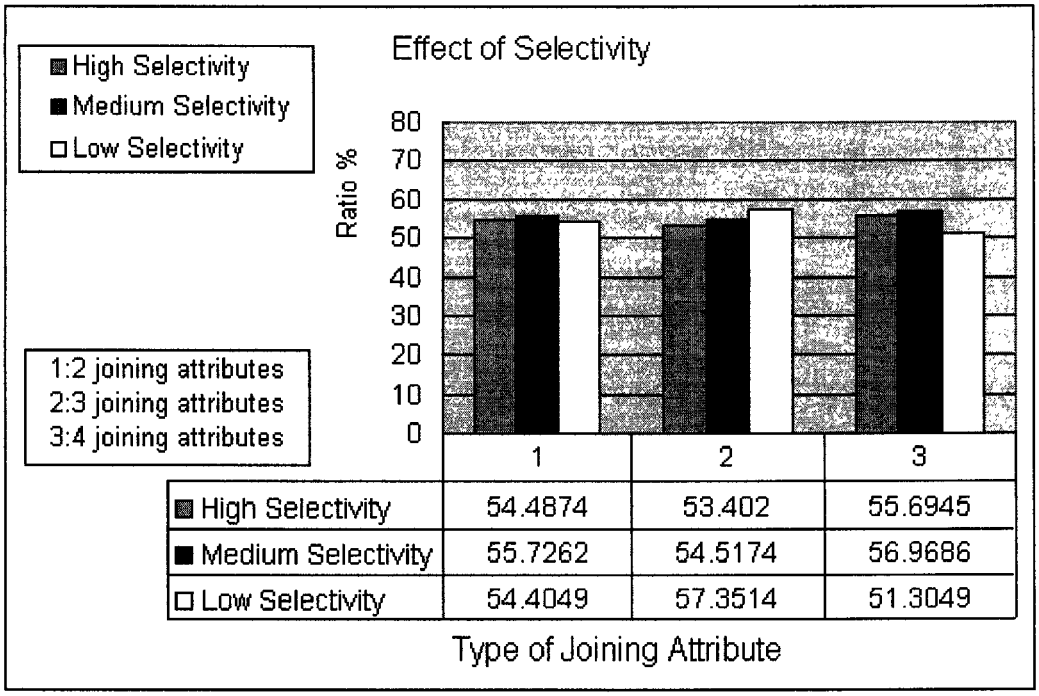

Figure 4.2.1.3 Five Relations

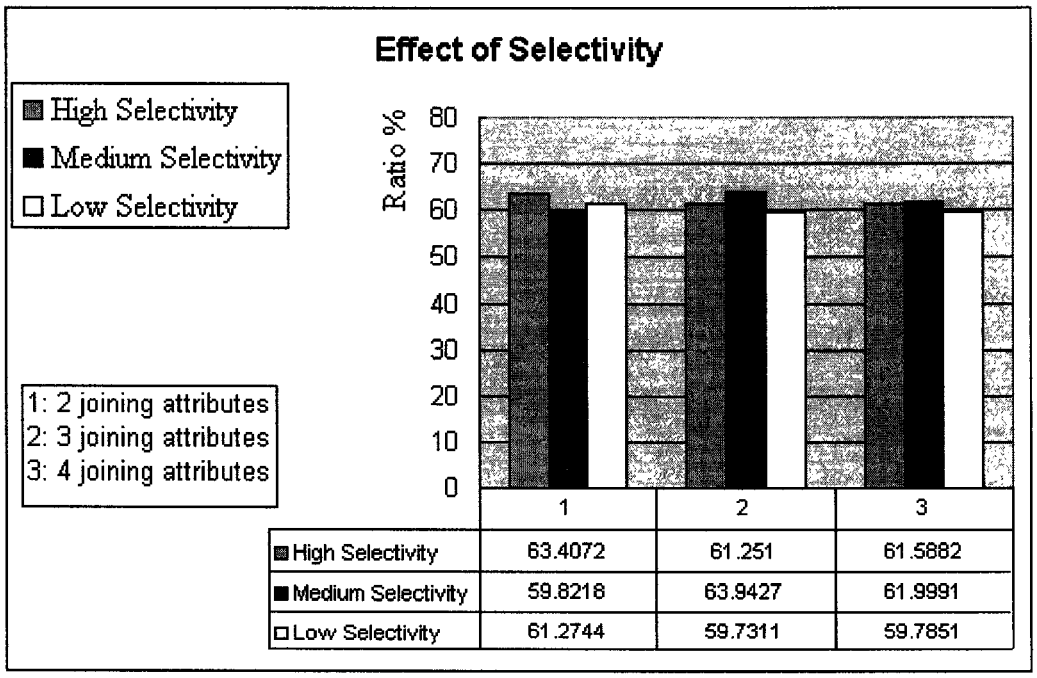

Figure 4.2.1.4 Six Relations

When we come to 5 relations or 6 relations, an outstanding phenomena is that more joining attributes don't necessarily worsen the performance, sometimes, such as 5

relations with high or medium selectivity, it benefits the whole performance. And compared to 3 relation or 4 relation cases, the beneficial rates are rising at large.

4.2.2 Effect of the number of joining attributes

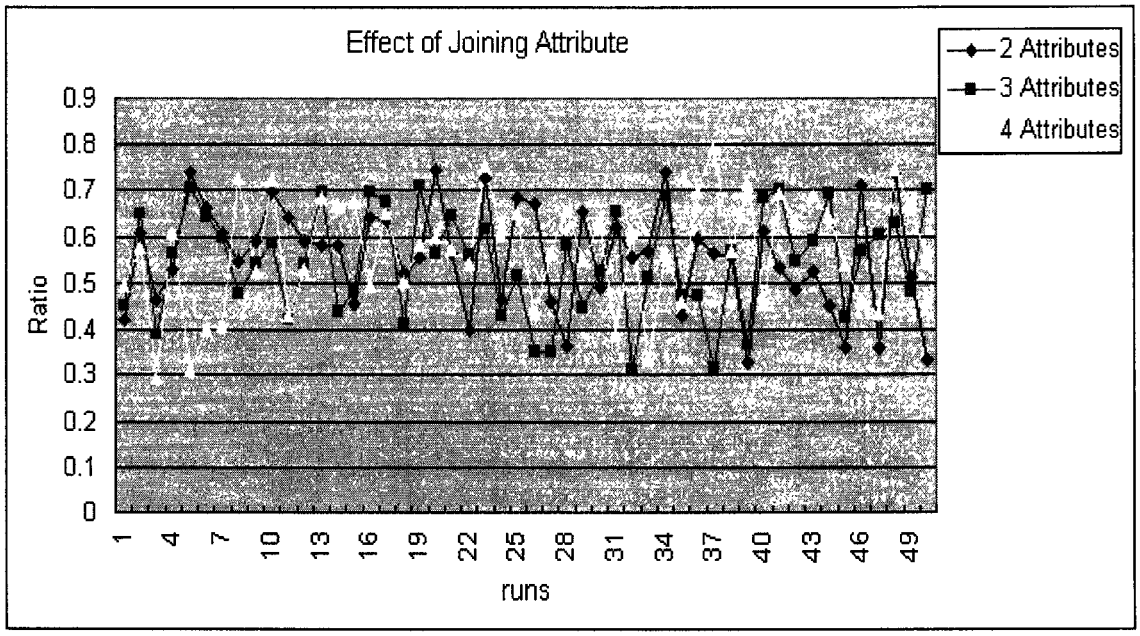

Figure 4.2.2.1 Joining Attributes' Effect on Five relations with Medium Selectivity

At a whole, when we are dealing with 4 joining attributes, the performance varies in a wider range. Sometimes it can give the relatively highest beneficial ratio, sometimes it can be the worst. For 2 joining attributes or 3 joining attributes, no special pattern found yet.

4.2.3 Effect of the selectivity

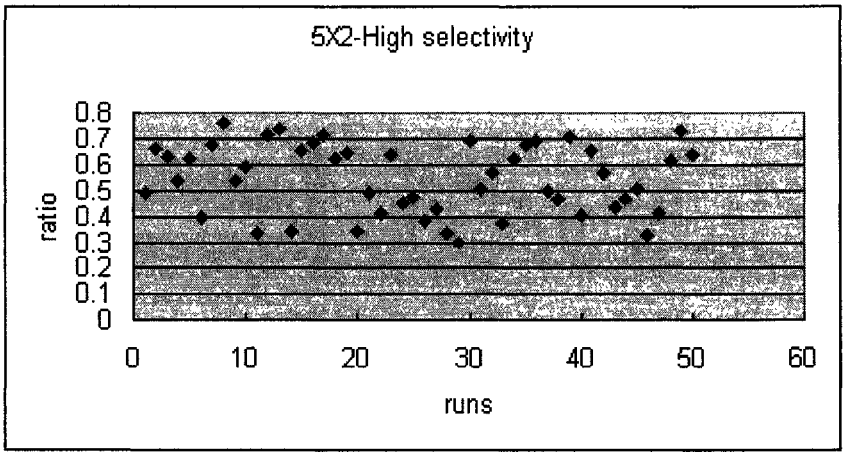

Figure 4.2.3.1 High Selectivity 5X2

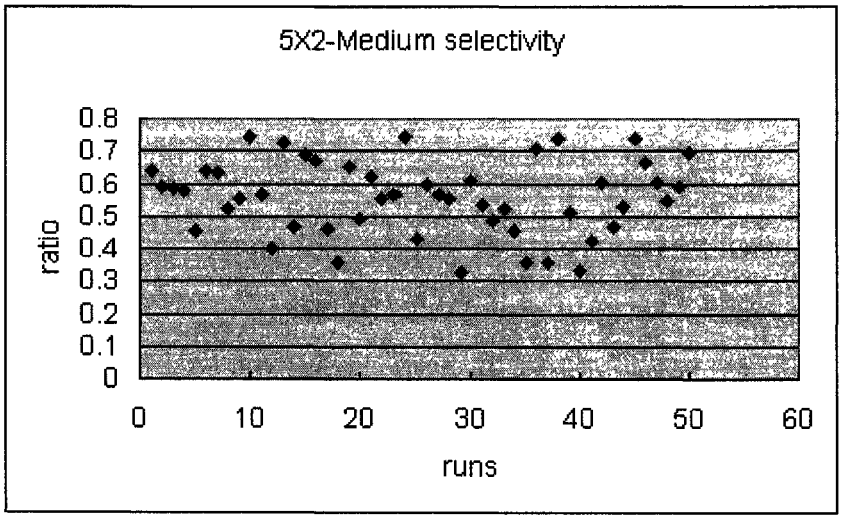

Figure 4.2.3.2 Medium Selectivity 5X2

Roughly speaking, higher selectivity is good for the performance when we have 5 relations for the simulation. The individual worst case occurs when we have low selectivity.

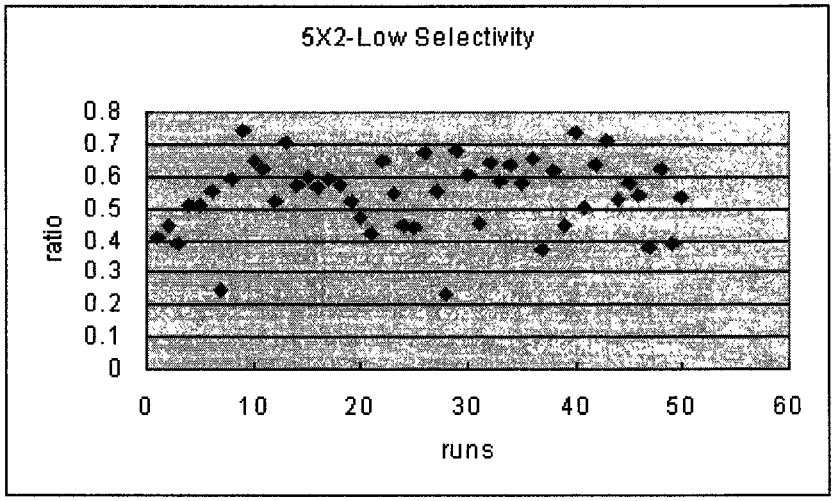

Figure 4.2.3.3 Low Selectivity 5X2

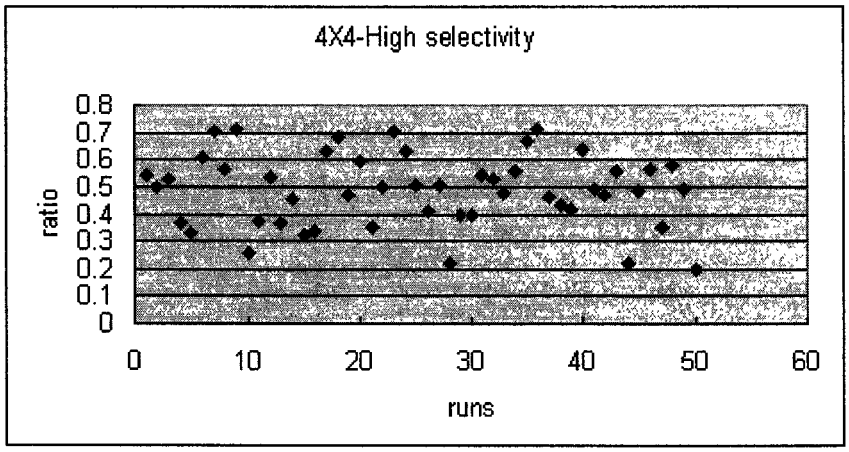

Figure 4.2.3.4 High Selectivity 4X4

However, when we come to 4 relations case, the situation changes a little bit. High selectivity is not bad, but medium selectivity can bring worst performance sometimes.

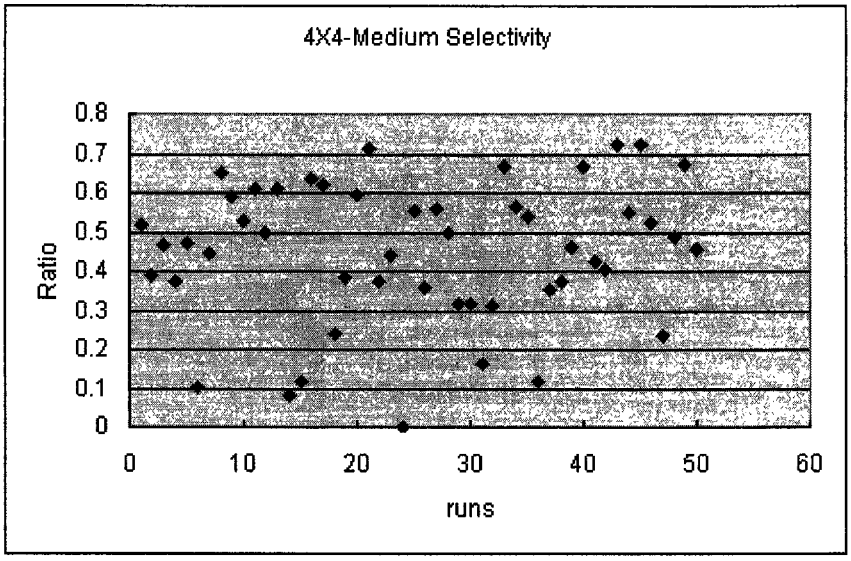

Figure 4.2.3.5 Medium Selectivity 4X4

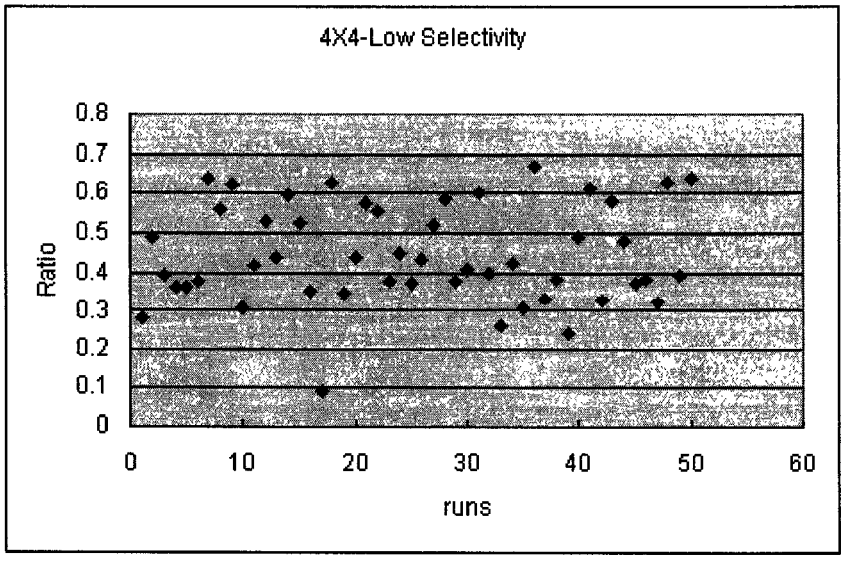

Figure 4.2.3.6 Low Selectivity 4X4

Just as shown in Figure 4.2.3.5, when we run the simulation with medium selectivity, sometimes, zero benefit can occur. But from all the experiments we did, we met no minus benefit, which means to some degree, this algorithm does not cause deterioration in performance.

4.2.4 Summary

Average beneficial ratio %			Number of relations					
			3			4		
			High	Medium	Low	High	Medium	Low
Number of joining attribute	2	U	57.971	65	63.8889	69.4915	69.2913	67.5926
		A	**31.5130**	**68.1790**	**64.2813**	**48.3794**	**47.0789**	**56.8182**
		L	0	0	0	9.7561	13.2353	7.9545
	3	U	64.375	64.1509	64.0625	72.2222	70.8333	69.7479
		A	**35.6417**	**69.6758**	**36.2892**	**46.5474**	**44.8458**	**46.6148**
		L	0	0	0	19.1919	7.8947	11.3821
	4	U	65.5172	60.7843	64.1509	71.1538	72.3164	66.6667
		A	**32.3129**	**34.3336**	**30.0733**	**48.7263**	**45.7247**	**44.4746**
		L	0	10.5263	0	19.7802	8.4507	9.0909
Average			**33.1559**	**57.3961**	**43.5479**	**47.8844**	**45.8831**	**49.3025**
Average beneficial ratio %			**5**			**6**		
			High	**Medium**	**Low**	**High**	**Medium**	**Low**
Number of joining attribute	2	U	76.4706	74.5665	74.1573	79.4286	80.0000	77.2727
		A	**54.4874**	**55.7262**	**54.4049**	**63.4072**	**61.2510**	**61.5882**
		L	30.0000	32.6389	23.1707	37.8698	42.0792	45.2703
	3	U	73.0435	72.0588	75.5274	78.6885	78.6517	79.5775
		A	**53.4020**	**54.5174**	**57.3514**	**59.8218**	**63.9427**	**61.9991**
		L	21.1538	30.9353	23.8806	38.5542	36.7647	36.2832
	4	U	74.5763	79.1667	72.9560	78.3069	80.9524	77.3946
		A	**55.6945**	**56.9686**	**51.3049**	**61.2744**	**59.7311**	**59.7851**
		L	23.2558	29.4643	16.3462	42.8571	35.9477	31.2500
Average			**54.5280**	**55.7374**	**54.3537**	**61.5011**	**61.6416**	**61.1241**

Figure 4.2.4.1 Average Benefit Ratio by factors

Above, we present results coming from the comparison between CRF and IFS. High, Medium and Low are the three categories for selectivity. U, A and L represent the upper, average and lower ratio.

So far we have finished the experiments that focus on the answer for our first concern: Is it beneficial to apply this algorithm on relations involved? The answer is positive, and we can say it's almost always beneficial and the benefit is on average between 30%~90%.

Next, we need to discuss the experiments that can outline the difference between bloom-filter algorithm and our algorithm CRF.

4.2.5 Comparison between bloom-filter and CRF

4.2.5.1 Full reduction

From reference [MOL00], we can get the average percentage of full reduction with collisions using a single set of filters or using two sets of filters in the simulating environment. This is summarized in the following tables:

Collision%	1	5	10	20	30	40	50	60	Average
3-2	26	19	19	18	19	17	21	13	21
4-2	50	41	40	34	39	29	29	14	37
5-2	70	66	58	47	43	44	40	30	54
6-2	80	66	52	46	41	36	32	25	52
Average	57	48	42	36	35	32	31	21	**41**

Figure 4.2.5.1.1 Average percentage of full reduction with collisions using a single set of filters

Collision%	1	5	10	20	30	40	50	60	Average
3-2	47	41	31	46	42	34	34	30	38
4-2	59	54	54	61	57	55	45	53	55
5-2	82	74	72	71	70	72	69	77	74
6-2	87	90	85	90	91	79	84	82	86
Average	69	65	61	67	65	60	58	61	63

Figure 4.2.5.1.2 Average percentage of full reduction with collisions using two sets of filters

In the above two tables, we know the first row represents the possible collision rate when using hash functions. The first column represents the types of queries, such as 3-2 represents 3 relations with 2 joining attributes. The data in the main area give information about the average percentage of full reduction under each collision situation of each type of query.

As we know, one of the good features of CRF is its full reduction capability. On this aspect, any other bloom-filter based algorithm can never do better than this algorithm. From the statistics of the above tables and from the past experimental data, even assuming the bloom-filter has a perfect hash function, the average full reduction percentage is only 74%. The above two tables show that, with a single set of filters, the full reduction rate can be only 41%. Even with two sets of filters, the rate is only raised to 63%.

4.2.5.2 Cost

On the other hand, we need to know the computational cost for each algorithm. So based on the above results from the bloom-filter based algorithm, we do the evaluation as follows:

1. Compare the result only using one single bloom-filter with CRF

Collision%	1	5	10	20	30	40	50	60	Average
3-2	89.27	86.65	87.56	87.10	84.44	78.52	80.39	70.12	83.59
4-2	94.73	94.29	94.63	92.88	92.43	90.49	87.88	86.35	91.92
5-2	97.65	97.22	98.58	97.80	96.53	96.06	93.57	92.23	96.42
6-2	99.14	98.97	99.43	98.57	98.58	97.86	97.73	94.23	98.17
Average	95.20	94.28	95.05	94.09	93.00	90.73	89.89	85.73	92.53

Figure 4.2.5.2.1 Average reduction with collisions using a single set of filters

On average, 92.53% of all relations involved are fully reduced by the bloom-filter based algorithm. There we just refresh the definition of average reduction (%) and full reduction (%).

Average reduction (%) = reduced size/(total size-full size)*100, where total size is the total size of all relations involved in queries, reduced size is the size of the relations being reduced by applying such algorithm, and full size should be the size the relations after being fully reduced.

For example, the total size of all relations involved is 10,000, the reduced size is 1,000, while the full size should be 8,000.

Then average reduction (%)=1000/(10000-8000)*100=50%

Full reduction is percentage of queries fully reduced out of the total queries. For example, we run each type of query for 50 times. Out of these 50 queries, 25 queries are fully reduced. So the full reduction (%)=50%.

2. Compare the result using two sets of filters with CRF

Collision%	1	5	10	20	30	40	50	60	Average
3-2	89.08	89.17	85.53	89.14	88.63	88.91	87.53	89.39	88.29
4-2	94.79	94.57	93.46	96.88	95.03	95.06	93.12	94.95	94.79
5-2	98.24	97.66	97.76	98.09	97.30	97.41	97.39	98.52	97.85
6-2	99.23	99.61	99.37	99.30	99.41	99.02	99.01	99.46	99.25
Average	95.38	95.25	94.03	95.85	95.09	95.11	94.26	95.58	95.05

Figure 4.2.5.2.2 Average reduction with collisions using two sets of filters

We can deduce from the two figures that the real average reduction rate, for only a single set of filters is (Average reduction) × (full reduction) = 41% × 92.53% = 37.9%, while for the two sets of filters, it is raised to 63% × 95.05% = 59.88%. Then the extra cost percentage we have to pay when using bloom filters would be: (100- 37.9)% = 62.1% for a single set of filters, and 1 - 59.88% = 40.12% for two sets of filters.

X-2 type query	3	4	5	6	Average
Average full reduction size	2696	5135	7400	9770	6250

Figure 4.2.5.2.3 Average full reduction size

From the experimental result I got based on CRF algorithm, the average beneficial rate for X-2 type (X here represents 3, 4,5 or 6 relations, 2 represents two joining attributes) query is 55.6%, which means the cost we have to pay using this is about 44.4%. Compared to the extra cost percentage we have to pay when using only one set of bloom filters, this is obviously better. When compared with the result from two sets of bloom filters, we can approximately see 40.12% (extra cost)+ ~ 5% (cost) = 45.12% is the real cost to be paid. Thus, CRF is a little bit more beneficial than using two sets of filters. In other words, CRF is comparable to two sets of bloom filters.

Evaluation

From the experimental results, especially referencing to the average beneficial rate (Figure 4.2.4.1), the general trends are:

1. This algorithm is almost always beneficial.
2. The lowest beneficial rate is 0 (occasionally occurs), and highest bound from the experiments we got is 81%. The average is 52.2%.
3. With the increase in the numbers of relations, the beneficial ratio is getting higher, however, the performance on three-relation type is exceptional.
4. Generally speaking, with high selectivity, our algorithm performs well, at least not worse than medium or low selectivity, except in situations in three relations.
5. Considering the non perfect hash function used for the bloom filter, collisions will always occur, which makes the result of the bloom filter algorithm not as good as its theoretical expectation. Based on the collision rate, we can see CRF works better than the bloom-filter based algorithm on average.

6. The past research only focuses on bloom-filters. When one set of bloom filters doesn't work well under collision situation, they found out using one more set of bloom filters could lead to better performance. From the experiments above we can see, even compared with the average result from two sets of bloom filters, CRF works alone better, at least not worse.
7. We conjecture that a bigger relation size will lower the benefit we can get using CRF. However from the experiments, it may not be true. Generally, when more relations are involved, the total size of the relations is bigger, but the performance is better. Another case we consider is the use of one set of bloom filters to reduce the relations, then use the CRF as the second step. In this experimental environment, we do not run under this circumstance. However, from the theoretical analysis based on the experimental result, it is not worse when using one set of bloom filters with CRF combination than using two sets of bloom filters.

Chapter 5 — Conclusion & Future work

5.1 Conclusion

The optimization of general queries in Distributed Database System (DDBS) is a very important research area, which is a core part in improving the whole database system's performance. The main concern or problem in this area is the selection of the best sequence of various operations to process queries to keep the cost to minimum. Because finding the optimal solution is NP-hard, a realistic and beneficial approach is to use heuristic algorithms which can produce near-optimal solutions.

During the past two decades, various possible algorithms have been presented and tested, which can be classified into following categories:

Join based algorithms, Semi-join based algorithms, hash-semijoin (or bloom filter) based algorithms, and join/semi-join combined algorithms. In general, semi-join based algorithms perform better than join-based algorithms. However what we can see is, we still have to spend a lot when using semi-join based algorithms for transmission. Focusing on transmission cost, not local processing cost, filters are proposed as a cheap way to implement semi-joins. This is a good approach compared to the former ones, 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. When collisions occur, we cannot reduce the relations to the full extent, which means a higher transmission cost than necessary. How can we overcome such a disadvantage of bloom filters? Some research proposed the use of more than one set of bloom filters. To a certain extent, it is better than only using a single set of bloom filter, for it reduces the

possibility of false-drop that occurs under collision. However, it still cannot promise the full reduction. It does give a way of solution, which is to increase the number of filters for use. However, a filter itself is not costless. It is not an easy job to find the balance between a false drop and amount of filters to use. So far no research has studied the formula or rules to find the right number of filters to use (Only in [Mul96], experimental numbers are given). In reality, there is more than one joining attribute in queries. Experiments show that using composite semi-join, we can get not worse result than semi-join. When taking all these factors into account, several considerations are very meaningful.

1. Replace semi-join with filters if possible.
2. Avoid collisions in filters if possible.
3. Apply composite semi-join into other algorithms if possible.

In this thesis, we investigate a new filter based algorithm CRF. This algorithm can use the filter concept. However it can avoid collisions. Thus after applying such algorithm, we can assure the full reduction. Secondly, it can keep the join information, which can combine composite semi-join with filters, however it's never possible for bloom filter-based algorithms.

Algorithm CRF is a filter-based algorithm, which gets insight from PERF join. The main procedure to implement CRF is as follows:

1. Send join attributes: Transmit join attributes projection(s) $P_{Ri'}$ in parallel to the assembling site. Combining the composite semi-join, join the $P_{Ri'}$ and generate a CRF(R_i') for each corresponding R_i'
2. Receive CRF: Ship CRF(R_i') to and combine with each R_i'. R_i' will be fully reduced into a relation R_i'' which contains only the matching tuples.
3. Transmit matching tuples: After excluding the join attributes, send each R_i'' in parallel to the assembling site to compose the final join result.

5.2 Future work

In my current work, I do not do any local actions for removal of duplicates. Even though we do not take local time into account, the local cost is still there. If doing the removal of duplicates on projections locally, sort and scan cost will occur. With the augmentation of relation size, the cost will be larger. So that in my current work, I do not remove the redundancy of projections, just send them for filter generation directly. However, we can tell, it will cost extra transmission cost for sure. Maybe future work can focus on finding out the real beneficial way to deal with the duplicates.

Secondly, due to the simulating environment for these experiments, I do not especially choose cases that favor the composite situation. But it's clear, if more special cases favor the composite situation, the result, beneficial rate will be higher. So future work can continue work on special cases to find out how beneficial the composite situation is.

Thirdly, the comparison between CRF and bloom-filter based algorithms is tested only limited to X-2 type queries. When joining attributes increase, combined with the situation of more composite joins occur, the test results need to be proved, though theoretically we expect CRF is better than bloom-filter based algorithms.

Fourthly, future work can further the research and put it into the real database product for performance testing and improvement if possible.

Reference

[AHY83]

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

[BFS00]

S.Bandyopadhyay, Q.Fu and A. Sengupta, "A cyclic multi-relation semijoin operation for query optimization in distributed databases", Proc. 19th IEEE International Performance, Computing and Communications Conference - IPCCC 2000 PERFORMANCE, February, 2000.

[BGW+81]

P. Bernstein, N. Goodman, E. Wong, C. Reeve, and J. Rothnie, "Query processing in a system for distributed databases (SDD-1)", ACM Transactions on Database Systems, Vol. 6(4), pp. 105 – 128, 1981.

[BMS96]

Bandyopadhyay, S; Morrissey, J; Sengupta, A, "Query optimization strategy for distributed databases on all-optical networks", Canadian conf. Electrical & computing Engineering, IEEE, Piscataway, NJ, (USA), vol. 1, pp. 245 – 248, 1996.

[BR88]

Peter Bodorik, J. Spruce Riordon, "Heuristic Algorithms for Distributed Query Processing", IEEE, pp.144 – 155, 1988.

[BRJ89]

Peter Bodorik, J. Spruce Riordon and C. Jacob, "Dynamic Distributed Query Processing Techniques", Proceedings of the seventeenth annual ACM conference on Computer science: Computing trends in the 1990's: Computing trends in the 1990's, pp. 348 – 357, February 1989, Louisville, Kentucky, United States.

[BRP92]

Peter Bodorik, J. Spruce Riordon, James S. Pyra, "Deciding to Correct Distributed Query Processing", IEEE Transactions on Knowledge and Data Engineering, Vol.4, No.3, pp. 253 – 265, June 1992.

[CA99]

Dunren Che and Karl Aberer, "A Heuristics-Based Approach to Query Optimization in Structured Document Databases", 1999 International Database Engineering and Applications Symposium, August 02 – 04, 1999, Montreal, Canada, p. 24.

[CCY92]

Tung-Shou Chen, Arbee L.P. Chen and Wei Pang Yang, "Hash-semijoin: A new technique to minimizing Distributed Query time", Proc. of the Third IEEE Workshop on Future Trends of Distributed Computing Systems, Taipei, Taiwan, R.O.C., April 1992.

[CL00]

Hao Chen and Chengwen Liu, "An Efficient Algorithm for Processing Distributed Queries Using Partition Dependency", Seventh International Conference on Parallel and Distributed Systems (ICPADS'00), July 04 - 07, 2000 Iwate, Japan, p. 339.

[CL84]

L.Chen and V.Li, "Improvement algorithms for semi-join query processing programs in distributed database systems", IEEE Transaction on Computers, Vol. 33(11), pp. 959 – 967, 1984.

[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.

[CMT+00]

Bogdan Czejdo, Ruth Miller, Malcolm Taylor and Marek Rusinkiewicz, "Distributed Processing of Queries for XML Documents in and Agent Based Information Retrieval System", Kyoto International Conference on Digital Libraries 2000, p.31.

[CR94]

Chungmin Melvin Chen and Mick Roussopoulos, "Adaptive Selectivity Estimization Using Query Feedback", ACM-SIGMOD International Conference on Management of Data, pp. 161 - 172, 1994.

[CY91]

Ming-Syan Chen and Philip S. Yu, "Determining Beneficial Semijoins for a join Sequence in Distributed Query Processing", ICDE 1991: pp. 50 – 58.

[CY92]

M. Chen and P. Yu, "Interleaving a Join Sequence with semi-joins in Distributed Query Processing", IEEE Transactions of Parallel and Distributed Systems, 3(5): pp. 611 – 621, September 1992.

[DF96]

Suzanne Wagner Dietrich & Changguan Fan, "An Application of Fragmentation Transparency in a Distributed Database System: A case study", Journal of Systems and Software, Vol. 35, No. 3, Dec. 1996, pp. 185-197

[GHR00]

Nalin Gupta, Jayant R. Haritsa and Maya Ramanath, "Distributed Query Processing on the Web", 16th International Conference on Data Engineering February 28 - March 03, 2000, San Diego, California, p. 84.

[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, Dec. 1995.

[Gra00]

Goetz Graefe, "Dynamic Query Evaluation Plans: Some Course Corrections?", IEEE Computer Society, June 2000, Vol.23, No.2, pp. 3 – 6.

[Gra96]

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

[GS96]

Bezalel Gavish and Arie Segev, "Set query optimization in distributed database systems", Vol. 11, No.3, pp. 265 – 293, 1986.

[GZ98]

Qadah, GZ, "Filter-based join algorithms on uni-processor and distributed memory multiprocessor database machines", Lecture notes, Vol.303, pp. 388 – 413, 1998.

[HCY94]

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

[HFC+00]

Joseph M. Hellerstein, Michael J. Franklin, Sirish Chandrasekaran and Amol Deshpande, "Adaptive Query Processing: Technology in Evolution", IEEE Computer Society, June 2000, Vol.23, No.2, pp. 7–18.

[HJ97]

Yun-Wu Huang and Ming Jing, "A Cost Model for Estimating the Performance of Spatial Joins Using R-trees", the 9th International Conference on Scientific and Statistical Database Management (SSDBM '97) August 11 – 13, 1997, Olympia, WA, p. 30.

[HKL94]

Jorng-Tzong Horng, Cheng-Yan Kao and Baw-Jhiune Liu, "A Genetic Algorithm for database query optimization", Proceedings of the First International Conference on Evolutionary Computation, pp. 350-355, 1994.

[HM00]

Abdelkader Hameurlain, Franck Morvan, "An Overview of Parallel Query Optimization in Relational Systems", 11th International Workshop on Database and Expert Systems Applications (DEXA'00), Sept. 06 - 08, 2000 Greenwich, London, UK, p. 629.

[HR94]

Evan P. Harris and Kotagiri R., "Join Algorithm Costs Revisited", VLDB Journal Vol.5, 1994, pp. 64 – 84.

[JKR99]

Vanja Josifovski, Timour Katchaounov and Tore Risch, "Optimizing Queries in Distributed and Composable Mediators", Information Systems, September 02 - 04, 1999, Edinburgh, Scotland, p. 291.

[JPS91]

Anant Jhingran, Sriram Padmanabhan and Ambuj Shatdal, "Join Query Optimization in Parallel Database Systems", SIGMOD Record 20(4), pp. 81 – 82, 1991.

[KG99]

Max Kremen & Jarek Gryz, "A survey of Query Optimization in Parallel Database", York University, November 1, 1999, Technical Report CS-1999-04.

[JS98]

Faiza Jajjar and Yahya Slimani, "Distributed Optimization of Cyclic Queries with Parallel Semijoins", DEXA Workshop 1998, pp. 717 – 722.

[KKM98]

Alfons Kemper, Donald Kossmann and Florian Matthes, SAP R/3 (tutorial): a database application system, p. 499,1998.

[Kos00]

Donald Kossmann, "The State of the Art in Distributed Query Processing", ACM Computing Surveys, Vol32, No.4, December 2000, pp.422 – 469.

[KR87]

H. Kang and N. Roussopoulos, "Using 2-way semi-joins in distributed query processing", in Processing 3rd International Conference on Data Engineering, pp. 644 – 650, 1987.

[KS00]

Donald Kossmann and Konrad Stoker, "Iterative Dynamic Programming" A new Class of Query Optimization Algorithms", ACM Transaction on Database Systems, Vol.25, No.1, March 2000, pp. 43 – 82.

[KTY82]

Larry Kerschberg, Peter D. Ting and S. Bing Yao, "Query optimization in star computer networks", pp. 678 – 711, 1982.

[Lia99]

Yan Liang, "Reduction of collisions in bloom filters during distributed query optimization", M.Sc. Thesis, University of Windsor, 1999.

[LC01]

Chang-Hung Lee and Ming-Syan Chen, "Distributed Query Processing in the Internet: Exploring Relation Replication and Network Characteristics", The 21st International Conference on Distributed Computing Systems, April 16 - 19, 2001, Mesa, AZ, p. 0439.

[LCK96]

Chenwen Liu, Hao Chen and Warren Krueger, "A Distributed Query Processing Strategy using Placement Dependency", ICDE 1996, pp. 477 – 484.

[Leg97]

Cesar Galindo-Legaria, "Outerjoin Simplification and Reordering for Query Optimization", ACM Transactions on Database Systems, Vol. 22, No. 1, March 1997, pp. 43 – 74.

[LOG93]

Hongjun Lu Beng-Chin Ooi and Cheng-Hian Goh, "Multidatabase Query Optimization: Issues and Solutions", Proceedings of Third International Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems, pp. 137-143, 1993.

[LOZ95]

Xuemin Lin, Maria E. O. and Xiaofang Zhou, "Using Parallel Semi-join Reduction to minimize Distributed query response time", Proceedings of the 1st IEEE International Conference on Algorithms and Architecture for Parallel Processing, pp. 517 – 526, April 1995, Brisbane, Australia.

[LPP91]

P. Legato, G. Paletta, and L. Palopoli, "Optimization of join strategies in distributed databases", Info. Systems, Vol. 16(4), pp. 363 – 374, 1991.

[LR94]

Li, Z., and Ross, K. A, "A new client-server architecture for distributed query processing", Tech. Rep. CUCS-014-94, Columbia University, 1994.

[LR95]

Zhe Li and Kenneth A. Ross, "PERF join: An Alternative to Two-way Semijoin and Bloomjoin", In CIKM '95, pp 137 – 144, 1995.

[LS91]

Hongjun Lu and Ming-Chien Shan, "On global Query Optimization in Multidatabase Systems", SIGMOD Conference 1991, pp. 168 – 177.

[LW86]

Stéphane Lafortune and Eugene Wong, "A state transition model for distributed query processing", ACM Transactions on Database Systems (TODS), Vol. 11 No.3, pp. 294 – 322, 1986.

[Ma97]

Xiaobo Ma, "The use of bloom filters to minimize response time in distributed query processing ", M.Sc. Thesis, University of Windsor, 1997.

[MB95]

J. Morrissey and S. Bandyophdhyay, "Computer Communication Technology and its effects on Distributed Query Optimization Strategies", Electrical and Computer Engineering, Canadian Conference, Vol.1, pp. 598 -601, Sep 1995.

[MIK+00]

Mena, Eduardo; Illarramendi, Arantza; Kashyap, Vipul; Sheth, Amit P, "OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies", Distributed and Parallel Databases, vol. 8, no. 2, pp. 223 – 271, 2000.

[MO98]

J. Morrissey and W. K. Osborn, "Distributed Query optimization using reduction filters", 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", Proceedings of the 1999 IEEE Canadian Conference on Electrical and Computer Engineering, May 1999.

[MOH84]

C.MOHAN, "Tutorial: Recent Advances in Distributed Data Base Management", ISBN 0-8186-0571-5, IEEE Catalog Number EH0218-8, IEEE Computer Society Press, 1984.

[MOL00]

J. Morrissey, Wendy Osborn and Y. Liang, "Collisions & reduction filters in distributed query processing", Can conf. Electrical & computing Engineering, IEEE 2000, vol. 1, pp. 240 – 244.

[Mul96]

James K. Mullin, "Estimating the size of a relational join", Information Systems, Vol. 18, No.3, pp. 189 – 196, 1993.

[NS99]

Faiza Najjar and Yahya Slimani, "Cardinality Estimation of Distributed Join Queries", Expert Systems Applications September 01 - 03, 1999 Florence, Italy, p. 66.

[NWM+99]

Kenneth W. Ng, Zhenghao Wang, Richard R. Muntz and Silvia Nittel, "Dynamic Query Re-Optimization", 11th International Conference on Scientific and Statistical Database Management, July 28 - 30, 1999 Cleveland, Ohio, p. 264.

[Ols94]

Michael Olson, "Mariposa: A new Architecture for Distributed Data", In Proc. 10th Int. Conf. on Data Engineering, pp. 54 - 65, Houston, Texas, 1994.

[OT99]

B.Johm Oommen and Murali Thiyagarajah, "Query Result Size Estimation Using a Novel Histogram-like Technique: The Rectangular Attribute Cardinality Map", 1999 International Database Engineering and Applications Symposium, August 02 - 04, 1999, Montreal, Canada, p. 3.

[OV91]

M.Tamer Ozsu and Patrick Valdureiz, "Distributed Database Systems: Where are we now? ", ACM Computing Survey, Vol. 24, No. 8, pp. 68-78, August 1991.

[OV99]

M. Tamer Ozsu & Patrick Valduriez, "Principles of Distributed Database Systems", Second Edition, Printice Hall, Upper saddle River, New Jersey 07458, 1999.

[Ozs00]

M.Tamer. Ozsu, "Next Generation Distributed DBMSs: Some Views & Research Directions", http://www.ualberta.ca/~ozsu.

[PC90]

William Perrizo, Chun-Shwu Chen, "Composite Semijoins in Distributed Query Processing", Information Sciences, Volume 50, No.3, April 1990.

[PCV93]

Jignesh M. Patel, Michael J. Carey and Mary K. Vernon, "Accurate Modeling of the Hybrid Hash Join Algorithm", SIGMOD Conference 1993, pp. 59 – 68.

[Pik97]

John Pike, "Distributed Characteristics and Performance Database (DCPDB)", http://www.fas.org/irp/program/disseminate/dcpdb.htm, 1997.

[PK01]

Sangwon Park and Hyoung-Joo Kim, "A New Query Processing Technique for XML Based on Sigmature", 7th International Conference on Database Systems for Advanced Applications (DASFAA '01), April 18 - 21, 2001 Hong Kong, China, p. 0022.

[PLL+99]

Ho-Hyum Park, Chan-Gun Lee, Yong-Ju Lee and Chin-Wan Chung, "Early Separation of Filter and Refinement Steps in Spatial Query Optimization", Proc. DASFAA, pp. 161 – 168, 1999.

[PRW94]

William Perrizo, Prabhu Ram and David Wenberg, "Distributed Join Processing Performance Evaluation", Proceedings of the Twenty-Seventh Annual Hawaii International Conference on System Sciences, Jan. 1994.

[PV88]

Sakti Pramanik and David Vineyard, "Optimizing join queries in distributed database", IEEE Transaction on software engineering Vol. 14, No.9, pp. 1319-1326, Sept. 1988.

[RBD96]

Fausto Rabitti, Leonardo Benedetti and Federico Demi, "Query Processing in Distributed PIOS", DEXA Workshop 1996, pp. 470 – 475.

[RK91]

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

[RP95]

Prabhu Ram and William Perrizo, "Multidatabase Global query optimization", proceeding of the 28th Annual Hawaii International Conference on System Sciences, p.253, 1995.

[SC97]

Scheuermann, Peter; Chong, Eugene Inseok, "Adaptive algorithms for join processing in distributed database systems", Distributed and Parallel Databases, vol. 5, no. 3, pp. 233 – 269, Jul 1997.

[SD89]

Donovan A. Schneider and David J. DeWitt, "A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment", Proc. ACM SIGMOD int'l. Conf. on Management of Data, ACM Press, NY, pp.110 – 121, 1989.

[Seg86]

Arie Segev, "Optimization of join operations in horizontally partitioned database systems", ACM Transactions on Database Systems Vol. 11 No.3, pp.48 – 80, 1986.

[SHC96]

Myra Spiliopoulou, Michael Hatzopoulos and Yannis Cotronis, "Parallel Optimization of Large Join Queries with Set Operators and Aggregates in a Parallel Environment Supporting Pipeline", IEEE Transactions on Knowledge and Data Engineering, pp. 429 – 445, June 1996.

[SKB+01]

Konrad Stocker, Donald Kossmann, Reinhard Braumand and Alfons Kemper, "Integrating Semi-join-Reducers into State-of-the-Art Query Processors", 17th International Conference on Data Engineering, April 02 – 06, 2001, Heidelberg, Germany.

[SMB+01]

Leonard Shapiro, David Maier, Paul Benninghoff and Keith Billings, "Exploiting Upper and Lower Bounds in Top-Down Query Optimization", 2001 International Database Engineering Applications Symposium (IDEAS '01), July 16 - 18, 2001, Grenoble, France, p. 0020.

[SMK97]

Michael Steinbrumm, Guido Moerkotte and Alfons Kemper, "Heuristic and Randomized Optimization for Join Ordering Problem", VLDB Journal 6(3): 191 – 208, 1997.

[SW91]

Dennis ShaSha and Tsong-Li Wang, "Optimizing Equijoin Queries in Distributed Databases Where Relations Are Hash Partitioned", ACM Transactions on Database Systems, Vol. 16, No. 2, pp. 279 – 308, June 1991.

[TC92]

Judy C.R.Tseng, Arbee L.P. Chen, "Improving Distributed Query Processing by Hash-Semijoin", Journal of Information Science and Engineering, pp. 525 – 540, December 1992.

[TC94]

Pauray S.M.Tsai and Arbee L.P. Chen, "Optimizing Entity Join Queries by Extended Semijoins in a Wide Area Multidatabase Environment", ICPADS 1994, pp. 676 – 681.

[TL95]

Kian-Lee Tan and Hongjun Lu, "Optimization of Multi-Join Queries in Shared-Nothing Systems", Journal of Computer Science and Technology, Vol.10, No. 2, March 1995, pp. 149 – 162.

[TR01]

David Taniar and J.Wenny Rahayu, "Parallel Processing of "GroupBy-Before-Join" Queries in Cluster Architecture", 1st International Symposium on Cluster Computing and the Grid, May 15 - 18, 2001, Brisbane, Australia, p.178.

[UFA98]

Tolga Urhan, Michael J. Franklin and Laurent Amsaleg, "Cost-Based Query Scrambling for Initial Delays", Proceedings of the 1998 ACM SIGMOD international conference on Management of data, 1998, Seattle, Washington, United States, pp. 130 – 141.

[Ull89]

Jeffrey D. Ullman, "Principles of Database and knowledge base Systems", Volume Π: The new Technologies, Standford University, 1989.

[VM89]

Bennet Vance and David Maier, "Rapid bushy join-order optimization with Cartesian products ", ACM SIGMOD, Vol.14, No.1, pp. 35 – 46, 1989.

[Wan80]

Chihping Wang, "The complexity of processing tree queries in distributed databases", SIGMOD Conference 1980, pp. 169 – 178.

[WC96]

Chihping Wang and Ming-Syan Chen, "On the Complexity of Distributed Query Optimization", IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 4, pp. 650-662, Aug. 1996.

[WLC91]

C. Wang, V. Li, and A. Chen, "Distributed query optimization by one-shot fixed precision semi-join execution", in Processing 7th International Conference on Data Engineering, 1991, pp.756 – 763.

[WWD+97]

L. Wang, M. Wing, C. Davis and N. Revell, "Query Processing and Optimization in Temporal Object-Oriented Databases", DASFAA 1997, pp. 381 – 390.

[WWH00]

Wang, Yijie; Wang, Yongjun; Hu, Shouren, "Parallel execution of multi-join query", Jisuanji Xuebao, vol. 23, no. 2, pp. 177 – 183, Feb. 2000.

[YL90]

Clement Yu and Chengwen Liu, "Experiences with distributed query processing", VLDB 1990, pp. 519 – 538.

[YL99]

Ramana Yerneni, Chen Li, "Optimizing Large Join Queries in Mediation Systems", SIGMOD Conference 1999, pp. 311 – 322.

Vita Auctoris

Name: Yue (Amber) Zhang
Place of Birth: Tianjin, P.R.China
Date of Birth: September 7, 1976

Education:

M.Sc. Computer Science
University of Windsor
Windsor, Ontario, Canada
2001 — 2003

B.Sc., Computer Science
Tianjin University
Tianjin, P.R.China
1994 — 1998

B.Sc., Management
Tianjin University
Tianjin, P.R.China
1994 — 1998

Honors and Awards:

OGSST 2002, 2003
University of Windsor Graduate Scholarship 2002

Working Experience:

Vice President of Graduate Student Society
University of Windsor
2002 — 2003

Graduate/Teaching Assistant
University of Windsor
2002 — 2003

Software Developer and tester
Microsoft and Symbiosys
1999 — 2001

Software Engineer
Long Computer Technology Co. Ltd.
1998 — 1999

Research Assistant
IBM-Tianjin University Computer Lab
1996 — 1998