An evaluation between Bloom Filter join and PERF join in Distributed Query Processing

Ming Pei
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
Pei, Ming, "An evaluation between Bloom Filter join and PERF join in Distributed Query Processing" (2008). Electronic Theses and Dissertations. 1004.
https://scholar.uwindsor.ca/etd/1004

This online database contains the full-text of PhD dissertations and Masters’ theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000 ext. 3208.
AN EVALUATION BETWEEN BLOOM FILTER JOIN AND PERF JOIN IN DISTRIBUTED QUERY PROCESSING

By
Ming Pei

A Thesis
Submitted to the Faculty of Graduate Studies 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
2007
© 2007 Ming Pei
NOTICE:
The author has granted a non-exclusive license allowing Library
and Archives Canada to reproduce, publish, archive, preserve, conserve,
communicate to the public by telecommunication or on the Internet,
loan, distribute and sell theses worldwide, for commercial or non-
commercial purposes, in microform, paper, electronic and/or any other
formats.

The author retains copyright ownership and moral rights in
this thesis. Neither the thesis nor substantial extracts from it
may be printed or otherwise reproduced without the author's
permission.

In compliance with the Canadian Privacy Act some supporting
forms may have been removed from this thesis.

While these forms may be included in the document page count,
their removal does not represent any loss of content from the
thesis.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives
Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public
par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans
le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique
et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse.
Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement
reproduits sans son autorisation.

Conformément à la loi canadienne sur la protection de la vie privée,
quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination,
il n'y aura aucun contenu manquant.
Abstract

Nowadays, with the explosion of information and the telecommunication era's coming, more and more huge applications encourage decentralization of data while accessing data from different sites [HFB00]. The process of retrieving data from different sites called Distributed Query Processing. The objective of distributed query optimization is to find the most cost-effective of executing query across the network [OV99].

Semijoin [BC81] [BG+81] is known as an effective operator to eliminate the tuples of a relation which are not contributive to a query. 2-way semijoin [KR87] is an extended version of semijoin which not only performs forward reduction like traditional semijoin does, but also provides backward reduction always in cost-effective way. Bloom Filter [B70] and PERF [LR95] are 2 filter based techniques which use a bit vector to represent of the original join attributes projection during the data transmission. Compare with generating a bit array with hash function in bloom filter, Perf join is based on the tuples scan order to avoid losing information caused by hash collision.

In the thesis, we will apply both bloom filter and perf on 2-way semijoin algorithms to reduce transmission cost of distributed queries. Performance of propose algorithms will compare against each others and IFS (Initial Feasible Solution) through amount of experiments.

Keywords:
Distributed Query Processing, Semijoin, Bloom Filter, Perf Join.
Dedication

To my family
Acknowledgements

First I would like to thank my supervisor, Dr. Joan Morrissey, for her academic advice, for her support and help throughout my master study. She has guided me in my whole master study, got me interested in the research, and carefully reviewed each update of the thesis. I would like to express my admiration for her dedication to teaching and research.

I would also like to thank my external reader, Dr. Yuntong Wang, my internal Reader, Dr. Jianguo Lu, and my thesis committee chair, Dr. Robert Kent for their great suggestions and valuable time given to review my thesis.

I would like to thank Dr. Liwu Li, for his guidance. His sudden and unexpected death was a shock to everyone. He is sincerely missed.

I also should give my thanks to my managers and colleagues in my work who gave me lots of encouragement during my thesis composition period.
# Table of Content

Abstract...........................................................................................................iii
Dedication........................................................................................................iv
Acknowledgements........................................................................................v
Table of Content .............................................................................................vi
List of Figures ..................................................................................................viii
Chapter 1 Introduction .....................................................................................1
Chapter 2 Background Review .........................................................................4
  2.1 Definition and Notation ...........................................................................4
  2.2 Cost Model ..............................................................................................4
  2.3 Join Operation .........................................................................................5
  2.4 Semijoin Operation ................................................................................6
    2.4.1 The SDD-1 Query-Processing Algorithm ........................................8
    2.4.2 The General Algorithm (AHY) ........................................................9
  2.5 DQP strategies based on semijoin ..........................................................11
    2.5.1 2-way semijoin ...............................................................................11
    2.5.2 Composite Semijoin .......................................................................13
    2.5.3 Domain-Specific Semijoin ...............................................................14
    2.5.4 Bloom Filter join (Hash Semijoin) ................................................15
    2.5.5 PERF Join .......................................................................................17
Chapter 3 Implementation of Algorithms ......................................................20
  3.1 Assumption .............................................................................................20
  3.2 Algorithms Description ..........................................................................21
    3.2.1 Algorithm Bloom Filter ................................................................21
    3.2.2 Algorithm PERF ............................................................................23
    3.2.3 Example of Proposed Algorithms ................................................24
Chapter 4 Experiment and Evaluation .........................................................29
  4.1 Methodology ...........................................................................................29
  4.2 Test Query and Platform ........................................................................30
    4.2.1 Test Query (query generator) .........................................................30
    4.2.2 Platform Implementation ...............................................................32
List of Figures

Figure 1: Example of Join ................................................................. 6
Figure 2: Example of IFS vs. Semijoin ................................................. 8
Figure 3: 2-way semijoin ................................................................. 12
Figure 4: hash semijoin ................................................................. 16
Figure 5: Example of PERF .............................................................. 18
Figure 6: Example of PERF .............................................................. 18
Figure 7: Table of a Query ............................................................... 25
Figure 8: Example XML of a Query .................................................. 32
Figure 9: Hierarchy of classes ......................................................... 33
Figure 10: Experiment Result of Algorithm IFS, BF and PERF ............. 35
Figure 11: Transmission Cost Comparison of IFS, BF and PERF ........... 36
Figure 12: Reduction Rate Comparison between BF and PERF .......... 37
Figure 13: Reduction Rate of PERF over BF .................................... 38
Figure 14: Reduction Rate with High Selectivity Level (0) .................... 39
Figure 15: Reduction Rate with Medium Selectivity Level (1) ............... 40
Figure 16: Reduction Rate with Low Selectivity Level (2) ..................... 41
Figure 17: Reduction Rate with Different Relation ............................. 42
Figure 18: Reduction Rate with Different Attribute ............................ 43
Figure 19: Reduction Rate with Different Domain Size ........................ 44
Chapter 1 Introduction

Database system once was built centralized to meet the needs of structured information. The increasing demand for efficient means of accessing data coupled with the need to manage increasingly large volumes of data has made distributed relational databases critically important in modern IT systems. Distributed relational database was brought into reality to achieve the advantages of performance, reliability, availability, and modularity.

A distributed database system is defined as a network which consists of processors (nodes) located dispersedly but interconnected to each other via communication channels [V02]. Distributed database is stored on several computers and each site varies in size and complexity. The sites connect to each other via network but self-maintained locally. An essential feature of distributed database is to allow users to access the data at the same time from geographically disperse locations and to retrieve target data set by means of queries.

Distributed Query Processing is the procedure that retrieves data from different sites [AHY83] [HF01]. To run a query in a distributed database, each site processes the query and returns the results to the final query site as an answer. Thus, query optimization becomes a major issue of distributed query processing. The objective is to find the most cost-effective way to execute query over the network. Typically, a given distributed query is processed in three phases as shown in [RK91] [KR87] [TC92] [BRP92] [BRJ87] [CL84]:
(1) **Local processing phase:** Selections and projections are performed at local nodes on the joining and target attribute.

(2) **Reduction phase:** A sequence of semijoins is used to reduce the size of a relation cost-effectively, with a result of a decrease cost of data transmission.

(3) **Final query processing phase:** at query site, the final query will be performed after all relations involved in the query are transmitted to this site.

Due to the local processing costs are negligible by comparing with the communication costs of data transmission. The principal challenge is to design and develop efficient query processing strategies to minimize the communication cost focus on the phase (2) and (3). Semijoin, who acts as powerful size reducer in phase 2, only transfers parts of relations during the distributed query processing against sending the whole relation as join does. Lots of heuristic algorithms are proposed based on semijoin.

As an extended version of semijoin, 2-way semijoin [KR87] not only performs forward reduction as the traditional semijoin operator does, but also provides backward reduction in an always cost-effective way. By using a bit array to represent the join attribute projection, filter technology can achieve the same result as a semijoin but at much lower cost. Two filter-based techniques which are bloom-filter join (Hash Semijoin) [TC92] and perf (Partially Encoded Record Filter) join [LR95] will be discussed in detail afterward. Bloom-filter uses a search filter which is generated with hash function to represent the semijoin projection, while perf join minimize the cost of backward reduction in 2-way semijoin by sending a scan ordered bit array. They have similar storage and transmission efficiency. However, with bloom-filter, semijoin may encounter losing join information as a result of hash collusion. Regardless of existing weakness, these 2 kinds of techniques are most powerful reducers with significantly cheaper transmission cost.
In this thesis, two filter based approaches will be implemented by applying Bloom Filter and PERF into a 2-way semijoin based algorithm. We will evaluate the reduction effect between two algorithms and IFS strategy by amount of experiments. The rest of thesis is organized as follows: Chapter 2 reviews the back ground of query optimization, several operators and core algorithms are presented; Chapter 3 does description of bloom-filter and perf join algorithm respectively; Chapter 4 evaluates the performance of two algorithms by experiments; In Chapter 5, we make final conclusion and future work.
Chapter 2 Background Review

2.1 Definition and Notation

In query processing, we need use the following notation and definition to construct and describe a query strategy:

**Projection:** The projection of relation R on a set of attributes A is denoted by R[A]. It is obtained by discarding all columns of relation R that are not in A and eliminating duplicated rows if necessary.

**Selection:** The selection of these tuples whose A-attribute values equal to a specified constant in relation R is denoted by R.A=constant (operators other than "=" e.g., ≥ and ≠ are also allowed)

**Benefit:** the data reduced by semijoin,

**Net benefit:** the value of benefit minus cost, if the net benefit is greater than 0, we call it is a cost-effective.

**Schedule:** the cost of data transmission used for reducing an involved relation and the transmission of the reduced relation to the query computer

2.2 Cost Model

In distributed query processing, query optimizer considers all the possible ways to execute a query and decides on the most efficient way based on the cost. Cost-based query processing assigns an estimated "cost" to each possible query plan, and chooses the plan with the smallest cost. Costs are used to estimate the runtime cost of evaluating the query. Total time and response time are two cost models used to measure the query execution plan. The total time cost model is the sum of the every single operation happened during the query processing; while the response time model is the elapsed time from the query initiation to the it's end. [AHY83]
The execution cost of a query involves both I/O and CPU spending for local processing at each participating site and communication cost between the networks. It is so happened that network transmission cost is relatively more significant comparing to local processing time. To simplify the problem, typically, local processing is considered negligible and transmission cost is stressed as the major concern. Data transmission cost from one site to another can be represented as a linear function:

\[ CT = T_{mrg} + T_{tr} \]

\( T_{mrg} \) is the fixed time for initialization, while \( T_{tr} \) is the time of data transmitted from one site to another.

### 2.3 Join Operation

Join, one of the essential operations, retrieves data from different site and relations. It is a common yet highly time-consuming query operation. As shown in Figure 1, given relation \( R_1 \) and relation \( R_2 \) on attribute \( A \), a join of \( R_1 \) and \( R_2 \) is denoted as \( R_1.A = R_2.A \) [YC84], where \( R_1 \) and \( R_2 \) are joining relations and \( A \) is the joining attribute. Join is obtained by concatenating each row of \( R_1 \) with each row of \( R_2 \) wherever the \( A \)-attribute values of the 2 rows are equal. Typically, in distributed system, it is very likely that the two relationships are not in the same site. In that case, sequence of operations will be applied to optimize queries. Usually the comparatively smaller sized relation will be transferred to remote site for join operation.
Figure 1: Example of Join  

However, some problems may rise during this processing. One is that result relation can be greater than total participating relations and therefore actually increase the cost instead of reducing it. Another problem is that network resources are largely wasted by transmission of an entire relation, while only a small percentage of tuples are required.

2.4 Semijoin Operation

As discussed in the previous chapter, some problems exist in join operator. Therefore, the theory of semijoin is introduced in late seventies. One of the objects of semijoin is to reduce inter-site communication cost. As a relation algebraic operation, semijoin selects a set of tuples in one relation that match one or more tuples of another relation on the joining domains in [BC81-1].

A semijoin from relation $R_1$ to relation $R_2$ on attribute $A$ is denoted by $R_1 \rightarrow_{A} R_2$, where $R_1$ is the sending relation, $R_2$ is the reduced relation, and $A$ is the joining attribute. It obtained by shipping $R_1[A]$ which is the projection over attribute $A$ of $R_1$ to the site where $R_2$ resides, then make join with $R_1[A]$ and $R_2$.

Here in the following instance, a semijoin from relation $R_i$ to $R_j$ on attribute $A$ is denoted as $R_i \rightarrow_{A} R_j$. Relation $R_i$ and $R_j$ resides on different sites respectively.
The result of this semijoin is the projection on the attributes of $R$ of the join of $R_i$ and $R_j$. There are 2 steps during one semijoin operation:

1. Send $R_i[A]$ from site $R_i$ to $R_j$

2. Reduce $R_j$ by eliminating tuples whose attribute $A$ are not matching any value in $R_j[A]$.

The cost of this semijoin is the size of projection which we denote as $s(R_i[A])$; the benefit is $s(R_j) - s(R_j')$ (suppose this semijoin reduces $R_j$ to $R_j'$). If a semijoin has benefit exceeding the cost we say it is a cost-effective semijoin.

To impress that semijoin operator acts as a size reducer in distributed query processing. We compare its efficiency with IFS (Initial Feasible Solution). IFS is defined as, for a given query, all the relations involved in the query from different site are retrieved and directly shipped to the query site where the join will be executed. As a basic and simple query processing, IFS will also be use to compare against the 2 proposed algorithms in the later section. The a) and b) of figure 1 shows the cost comparison between semijoin and IFS. We can save 6 units cost during the transmission by replacing the IFS with semijoin.
AHY [AHY83] and SDD-1 [BC81] are two important algorithms based on semijoin. Both of them are under the assumption that at least one copy of each relation participated in the query has been chosen, then involved the reduction and assembled to the final query site.

2.4.1 The SDD-1 Query-Processing Algorithm

Algorithm SDD-1 [BC81] is the first technique based on semijoin in distributed query processing. Under the assumption of the network bandwidth being the bottleneck, the object of SDD1 Algorithm is to process the queries with a cost of inter-site data transfer as minimized as possible.

Three essential phases are involved in SDD-1 processing:

The first phase: Translate an adjective language query Q into a relational calculus form, known as an envelope, which can specify a data set as the result to Q.
The second phase: Construct relational operations called reducer P and select a site S so that the cost of compiling P and transferring results to S is the minimum between all reducers and sites. The major issue of this phase is using a greedy optimization algorithm to derive the sequence of semijoins that will retrieve the set of data needed for the query.

The third phase: Finish the local processing of Q from phase 2 at Site S. The execution in this phase uses the data from the 2nd phase and only involves local computing, therefore not discussed further in this section.

Although SDD-1 is an optimization algorithm, it still has few drawbacks and limitations. Because the selecting semijoins maximize immediate gain due to SDD-1 only pick up local optimal strategies. It will ignore the higher-cost semijoin which would result in increasing the benefits and decreasing the costs of other semijoins at each step of the strategy. Therefore, this algorithm may not be able to select the global minimum cost solution.

2.4.2 The General Algorithm (AHY)

In 1983, Apers, Hevner and Yao devised Algorithm General (Algorithm AHY) [AHY83] by improving Algorithm Serial and Parallel to deal with general query in distributed database. They figured out that it is cheaper to complete final join after relations are reduced by semijoin. They identified optimization objectives as the minimization of either response time or total execution time.

Algorithm General are constructed by following 4 phases
1) Local processing using selection and projection
2) Generate candidate relation schedules, consider each of σ join attributes a simple query
   a) Compute minimal response time for each join attribute by applying algorithm Parallel to each simple query.
   b) Compute minimal total time by applying algorithm Serial to each schedule.
3) Integrate the candidate schedules by doing procedures Response, Total and Collective.

4) Remove schedule redundancies, if necessary

There are three versions of procedures for Algorithm General. Procedure Response is used to reduce response time, while Procedure Total and Collective are used to minimize the total time.

A. Procedure Response

1) Candidate Schedule Generating and Ordering: For each relation Rij, generate candidate schedules on joining attribute dij, j = 1, 2, ..., σ. Sort these candidate schedules in ascending order of arrival time.

2) Schedule Integration: For each candidate schedule in ascending order, construct an integrated schedule for Rij that consists of the parallel transmission of candidate schedule, and select the integrated schedule with minimum response time.

B. Procedure Total

1) Generating Candidate Schedules: For each schedule containing a transmission of a join attribute dij, add another candidate schedule to minimize the total time:

2) Select the best candidate schedule: Select BESTij with minimal total time of transmitting relation Rij.

3) Order the candidate schedule: BESTij on joining attribute dij, j=1, 2, ..., σ, so that ART1 + C (s1 * SLT1) ≤ ... ≤ ARTσ + C (sσ * SLTσ), SLTij defines the accumulated attribute selectivity of the BESTij into Rij.

4) Integrate Schedules: Upon each candidate schedule BESTij, j=1, 2, ..., σ, an integrated schedule will be formed for relation Rij. Select the integrated schedule that results in the minimum total time value.
C. Procedure Collective
Because procedure Total does not consider the existence of redundant data transmissions in generating all candidate schedules, algorithm General may not be optimal. Procedure Collective is used to remove the most costly data transmissions.

Algorithm AHY (General) is an efficient algorithm which derives close to optimal query processing strategy. It can be applied to any general distributed query environment.

2.5 DQP strategies based on semijoin
To minimize the transmission overhead in most cost-effective way, researchers derives lots of algorithms and techniques based on semijoin to deal with variant circumstance. The following section will introduce some of them.

2.5.1 2-way semijoin
Kang and Roussopoulos proposed an extended version of semijoin which called 2-way semijoin [KR87]. A 2-way semijoin between 2 relations R_i and R_j over a attribute A can be denoted as t: R_i<-A->R_j, or { s: R_i->A->R_j, s': R_j->A->R_i}. This extended semijoin is used to reduce the size of both relations R_i and R_j for the final processing in 2 directions:

**Forward processing s:** relation R_i is first projected on join attribute (a_ij), noted as R_i |A|, and is sent to R_j to reduce the size of R_j by eliminating tuples whose attribute does not match to R_i |A|.

**Backward processing s':** During the forward process, R_i |A| is divided into R_i |A|m and R_i |A|nm. R_i |A|m is a set of values in R_i |A| that have match in R_j |A| and R_i |A|nm is R_i |A|- R_i |A|m. R_i |A|m or R_i |A|nm, whichever is less, will be sent back to R_i to reduce its size.
The cost of a 2-way semijoin is not as simple as \( C(s) + C(s') \). In most cases, the cost shall be computed as \( s(R[A]) + s*\min[R_i[A], R_mA] \). \( C(s) + C(s') \) is valid only if \( s' \) is delayed until \( s \) is finished, because then \( R_j[A] = R_i[A] \). And the benefit of a 2-way semijoin is the sum of benefits of \( s \) and \( s' \): \[ [s(R) - s(R')] + [s(R_j) - s(R_j')] \]. Figure 2 shows how the 2-way semijoin works.

**2-Way Semijoin**

![2-Way Semijoin Diagram](image)

Figure 3: 2-way semijoin

It is known that an extended semijoin shall replace the tradition semijoin if it is proved more cost effective. Comparisons of reduction power effects between 2-way semijoin and traditional semijoin is given in [KR87].

When a 2-way semijoin reduce Relation \( R_i \) to \( R_i' \), Cost of this 2-way semijoin for relation \( R_i \) is \( C_{tw} = w*|s_j| * \min[R_i[A], R_i[A]_m] \); Benefit of this 2-way semijoin for relation \( R_i \) is \( B_{tw} = w*|R_i| - |R_i'| \). It is observed that \( w \geq w*|s_j| \) and \( |R_i| - |R_i'| \) \geq \min[R_i[A]_m, R_i[A]_m], therefore, net value of this 2-way semijoin \( D_{tw} = B_{tw} - C_{tw} \geq 0 \). A semijoin \( R_i-\text{A}\rightarrow R_j \) is cost effective, while the 2 way semijoin \( R_i\leftarrow\text{A}\rightarrow R_j \) is cost
effective as well. It is concluded that 2-way semijoin is always cost-effective no matter whether only one or none can be reduced cost–effectively using traditional semijoin.

Based on the study on 2-way semijoin, they developed a new join algorithm for the purpose of reducing I/O cost, a n-way pipeline algorithm. [RK91] The main goal of this pipeline algorithm is to eliminate the needs of shipping, storing, and retrieving foreign relations and(or) intermediate results on the local disks of the query site during the processing of a join, even an N-way join.

2.5.2 Composite Semijoin

Composite semijoin was proposed to deal with situations where multiple columns are involved in projection and transmission [PC90]. Typically a processing algorithm will generate numerous semijoins preformed with common source and common destination sites. However, in a situation like this, it may be of more assistance to perform semijoins as one composite against as several single sites. The authors demonstrate the possible enhancement of two classic semijoin algorithms applying composite semijoin. One is variations on Algorithm General response-time version [AHY83], using selectivity as major estimation scheme. The other is variations on Algorithm W, using “worst case elimination” as the measurement of attribute size after semijoin. Experimental results reveal that composite semijoin typically will be more beneficial against common semijoin up to 24%.

Composite semijoin is not always a superior strategy. Sometimes it may produce higher response time. Hypothetically, composite semijoin results at least as good performance if not better. This is more likely to be true if composite semijoin is replacing a parallel semijoin not a serial one. Therefore it is safe to say that it is a better approach combining semijoin and composite semijoin.
2.5.3 Domain-Specific Semijoin

In the distribute database system, many query optimization algorithms proposed for fragmented databases apply semijoins to reduce the size of the fragments of joining relations, then send the resulting to the final processing site. While the traditional semijoin can not process 2 fragments due to it may eliminate tuples before they are compared with all tuples of other joining relations. Chen and Li devise a new semijoin operator named domain-special semijoin which can be performed in a fragment-to-fragment manner [CL90]. It exploits the semantic information associated with the joining fragmented relations and provides more flexibility. As a query strategy we often use both domain-specific semijoins and semijoins. They work together can guarantee the reduction effect at least as good as the best way with only semijoin reduction.

We define the domain-specific semijoin as follows:
\[ R_{ik} (A = B) R_{jm} = \{ r | r \sqcap R_i \cap A \sqcap R_{jm} \cap B \subseteq (\text{Dom}(R_{ik}.B) - \text{Dom}(R_{jm}.B)) \} \]

Where A, B are the joining attributes, Rik and Rjm are two fragments of the joining relations Ri and Rj respectively. Compare with running semijoin in the horizontally partitioned database from the fig listed blow:

To estimate the size of intermediate query processing result, let Rik' be \( R_{ik} (A = B) R_{jm} \). According to the definition of domain-specific semijoin, the number of tuples reduced by \( R_{ik} (A = B) R_{jm} \) is given by:
\[ |Rik| - |Rik'| = |Rik| \left( \frac{|\text{Dom}(R_{ik}.A) \cap \text{Dom}(R_{jm}.B)|}{|\text{Dom}(R_{ik}.A)|} \right) (1 - \left( \frac{|R_{jm}|}{|\text{Dom}(R_{jm}.B)|} \right)). \]

The benefit of \( R_{ik} (A = B) R_{jm} \) is \( \text{Ctran}(|R_{ik}| - |R_{ik}'|)w(R_i) \), where \( w(R_i) \) is the unit transmission cost and \( w(R_i) \) is the width of \( R_i \).

The cost of \( R_{ik} (A = B) R_{jm} \) is \( \text{Ctran}(|R_{jm}|) w(R_{jm}.B) \).

With the estimation we can perform domain-specific semijoin with fragmented relations by following steps:
(1) Calculate its estimated benefit and cost according to the formulas presented in the previous section.

(2) If it is found to be profitable, include it in the current query-processing strategy; otherwise, ignore it.

(3) Update the related information in the database profile according to the suggested formulas in the previous section if the domain-specific semijoin is included in the current strategy.

Because domain-specific semijoin operation not only takes advantage of fragmentation design but also avoid unnecessary processing. We can get that for a given query, there is always a strategy using both domain-specific semijoins and semijoins in the fragment–fragment manner. It is at least as good as the best strategy than using semijoin reductions only.

### 2.5.4 Bloom Filter join (Hash Semijoin)

Tseng and Chen introduce hash semijoin as a cost saving semijoin operator in [TC92].

In a hash semijoin, also called bloom filter join, a search filter, which can be viewed as an array of bits, is transmitted between relations instead of the semijoins’ projection. Initially, all bits in the array are set to 0. d hash functions hash each value in the projection into d bit addresses, setting each of the d bits to 1. Same hash function applied to the values of join attributes in apprentice relation R_j and generate another sequence of bit addresses. If all these d bits are 1 in the array, tuples in R_j containing this value will be selected as a semijoin result. R_i \bowtie R_j is denoted as a hash-semijoin of R_i and R_j. Based on the assumption, for a specific F, the size of the bit array, the search filter is optimal when the bit array is half full of 1 bits, F = (d/ln2)|R_j|.

However, the array of the bits may not be an accurate semijoin results, because information may lose in representing a value with bits. If a bit is set to 1, it is
either the attribute actually presented or a different attribute falsely dropped due to a collision. The possibility that a value is falsely accepted by the search filter is known as a false drop, \( f \), and the false drop probability is \( f = (1/2)^d \). Then the net benefit of the hash semijoin is \( BH_{ij} = (1-s_{ij}-f)w_j|R_j| - (d/\ln2)|R_i| \). A hash semijoin program shall replace a tradition semijoin program if \( BH_{ij} - BT_{ij} - fC_j > 0 \), as it is more cost-effective. Figure 3 shows how the hash semijoin works.

**Bloom Filter Join**

Tseng and Chen's define that a semijoin program is more cost-effective if its summation of the potential costs of all the semijoins contained is less. The potential cost of a semijoin operation is defined as the total of the cost of executing the semijoin operation and the cost of transmitting the result of the semijoin operation. Upon such assumption, a backward replacement algorithm, to replace traditional semijoin with hash semijoin, therefore is formed. The replacement algorithm works backward through the execution tree of a
distributed query, from the leave nodes to the root node. The mechanism of this algorithm works as follows:

Step 1: Identify nodes in the execution tree, sort and mark a node by its level
Step 2: Let the lowest node be $R_j$, the direct predecessor of this node be $R_i$, $R_i \rightarrow A \rightarrow R_j$.
Step 3: If $R_i \neq \text{nil}$, replace the traditional semijoin between $R_i$ and $R_j$ with a hash semijoin if $CT_{ij} - CH_{ij} - fC_j > 0$ and accumulate $CH_{ij}$ and $(s_{ij} + f)C_j$ to the potential cost of $R_i$, otherwise accumulate $CT_{ij}$ and $s_{ij}C_j$
Step 4: Apply step 3 to next lowest node and end the process at the highest level.

This algorithm assumes that the semijoin program contains only traditional semijoins, which is represented by an execution tree with each node having only one direct predecessor.

Dr. Morrissey and her colleagues find that the combination of semijoins and hash-semijoin [MO99] can make better performance than use semijoin only. They also improve that the collision caused by hash function in the filter does not have a huge impact on the performance even the collision rate at 50% [MOL00].

2.5.5 PERF Join

PERF join is a novel 2-way join presented by Li and Ross [LR95]. It is designed to minimize the transmission cost during the backward phase of 2-way join.

The acronym of PERF is "Positionally Encoded Record Filters". It is based on physically tuple scan order fashion. Suppose there is a join between 2 relations R and S. the PERF is a bit vector with number of n bit (n = cardinality of relation R) which is used to represent the join information of relation S. The jth bit of the vector will set to 1 only if the jth tuple of R appears in the join result. The following figure shows the PERF for Relation R over the join with S.
The basic idea of PERF join is mainly based on the 2-way semijoin. In 2-way semijoin, join attribute is projected on relation $R_i$, and is sent to $R_j$ to reduce the size of $R_j$. During the process, $R_j |A|_m$ or $R_j |A|_{nm}$, which is less, will be sent back to $R_i$ to reduce its size. Instead of transmitting $R_j |A|_m$ or $R_j |A|_{nm}$ back to relation $R_i$, PERF join sends a bit vector that contains one bit for every tuple in $R_i |A|$.

Figure 5: Example of PERF

**PERF Join**

![Diagram of PERF Join]

**Figure 6: Example of PERF**

Cost = 8 + 4 bits
It is known that PERF bit vector is significantly smaller in size than the 2-way semijoin result $R_i |A|_m$ or $R_i |A|_{nm}$, and at least has the same storage and transmission efficiency as a hash filter. Moreover, PERF is based on the tuple scan order, the order of bits in the bits vector which is the same tuple order of $R_i |A|$ that $R_i$ initially sent. Therefore, the PERF join does not suffer any loss of join information incurred by hash collision. Another observation is that PERF join is generated after applying a real join, it shall carry complete join information and is thereby able to handle more complicated and inequality join queries, such as cyclic join queries.
Chapter 3 Implementation of Algorithms

In this chapter, we present perf join and bloom filter algorithms based on 2 way semijoin algorithm [535N] [Y2005] by replacing the join attributes projection with perf and bloom filter respectively.

3.1 Assumption

The algorithms we proposed are based on the following assumptions.

1) We assume the relational data in the Distributed Database Management System has no fragmentation or replication.

2) Only select-project-join (SPJ) query is considered. There is no set operations like UNION, INTERSECTION, PRODUCT, and DIFFERENCE involved in the research.

3) A query consists of a number of relations, each of them reside at different site, and the result made available at the query site. Each relation can have a number of join attributes.

4) We assume the cost model is \( C(X) = C_0 + X \), where \( C_0 = 0 \) for simplicity. \( X \) is the amount of data transmitted.

5) We assume that we have a perfect hash function which the filter size is the same as the domain size.

6) When a semijoin operation occurred between 2 relations. Only the currently involved join attributed size will be reduced while other attributes properties will has no change.
3.2 Algorithms Description

The basic idea of our proposed algorithm is use bloom filter and perf filter to instead of the original semijoin projection during the procedure of data transmission.

3.2.1 Algorithm Bloom Filter

In this algorithm, we will apply bloom filter both in forward reduction phase and backward reduction phase. In the forward reduction phase, we construct the bloom filter for join attribute projection. While in the backward reduction phase, we need to build a bloom filter for the join match set. The detail steps of the algorithm are listed below.

Steps of Algorithm BF:

1. Arrange all relations by size in ascending order such that \( S(R_i) \leq S(R_j) \leq S(R_k) \leq \ldots \leq S(R_m) \).

2. For each joining attribute, we get a list of relations which contain this attribute by the order of step 1.

3. Generate an execution schedule \( R_i \to R_j \to R_k \ldots R_m \to R_i \) for each of joining attribute. \( R_i \to R_j \to R_k \ldots R_m \) is forward reduction while \( R_m \to R_{m-1} \to \ldots R_k \to R_j \to R_i \) we call backward reduction.

4. Start to execute schedules. The schedule with the smallest first projection size will be executed first.

5. For each predicate between the 2 relations in forward reduction phase of the schedule. We generate the bloom filter which has the same bit number with domain size base on our assumption. If the bloom filter size is smaller than projection size and net benefit is greater than 0, we execute this
bloom filter join by sending the bloom filter form $R_i$ to $R_j$. Otherwise we ignore it and execute the bloom filter join between next 2 relations over the join attribute. For example, if the net benefit of $R_i \rightarrow R_j \leq 0$ or $BF(R_i[A]) > |R_i[A]|$, we ignore this and execute $R_i \rightarrow R_k$.

6. Once a semijoin has been done in forward reduction phase. We need to record it in an arraylist which we called semijoin list. And also keep the join match set and non match set in a data structure.

7. Start backward reduction follow the schedule. For relation $R_j$, $R_i$ over join attribute $A$, if it exists in the in bloom filter execution list, it means the bloom filter join from $R_i$ to $R_j$ on attribute $A$ has been done already in the forward reduction phase, thus, we could find out the match set ($m\_set$) or non match set ($nm\_set$) for $R_i[A]$ of relation $R_j$ and generate another bloom filter for either match set or none match set due to both of them will have the same size in bloom filter. Compare the size of bloom filter for the match set and $\min\{m\_set, nm\_set\}$. send the smaller one back to $R_i$ to reduce $R_i$. The transmission cost is the size of the smaller one which has been sent. If there is no record of relation $R_j$, $R_i$ over join attribute in the executed bloom filter join list which means there was no bloom filter join executed from $R_i$ to $R_j$ on attribute $A$. so we need to consider to do the bloom filter by the term of net benefit $>0$ and also the bloom filter size is smaller than projection size.

8. Repeat steps 4,5,6,7 till all of schedules have been processed.

9. Output the transmission cost which is the sum of every reduced relation size plus the transmission cost in every single schedules.
3.2.2 Algorithm PERF

Compare with 2-way Semijoin, PERF join is has better performance if and only if 
1<\(W(A) \times \min(p(R), 1-p(R))\). It means that PERF join can not exceed 2-way 
semijoin all the time. There still exists chance for 2-way semijoin could have less 
transmission cost than PERF join when \(\min(p(R), 1-p(R))\) is very low. To make 
better efficiency, we apply one of them which have lower cost into the back 
reduction phase of 2-way semijoin algorithm. Therefore the new algorithm could 
perform at least as the old one. The core of this algorithm is based on Algorithm 
UPSJ [Yang05]. The detail steps of the algorithm are listed below.

Steps of Algorithm PERF:

1. Arrange all relations by size in ascending order such that \(S(R_i) \leq S(R_j) \leq S(R_k) \leq \ldots \leq R_m\).

2. For each joining attribute, we get a list of relations which contain this 
   attribute follow the order from step 1.

3. Generate an execution schedule \(R_i \rightarrow R_j \rightarrow R_k \ldots \rightarrow R_m \rightarrow R_j \rightarrow R_i\) for each 
   of joining attribute. \(R_i \rightarrow R_j \rightarrow R_k \ldots \rightarrow R_m\) is forward reduction while \(R_m \ldots \rightarrow R_k \rightarrow R_j \rightarrow R_i\) is backward reduction.

4. Start to execute schedules. The schedule with the smallest first projection 
   size will be executed first.

5. For each predicate between the 2 relations in forward reduction phase of 
   the schedule, only execute the semijoin by the term if the net benefit is 
   greater than 0. Otherwise execute the next predicate. For example, if the 
   net benefit of \(R_i \rightarrow R_j \leq 0\), we ignore this and execute \(R_i \rightarrow R_k\)
6. Once a semijoin has been done in forward reduction phase. We need to record it in an arraylist which we called semijoin list. And also keep the join match set and non match set in a data structure.

7. Start backward reduction follow the schedule. For a semijoin between relation \( R_i \), \( R_j \) over join attribute \( A \), if it exists in the semijoin list, it means the semijoin from \( R_i \) to \( R_j \) on attribute \( A \) has been done already during the forward reduction phase. Thus, we could generate the bit vector \( \text{PERF}(R_i) \) for the projection of \( R_i[A] \). while we could also find out the match set(\( m\_set \)) and non match set(\( nm\_set \)) for \( R_i[A] \) of relation \( R_j \); compare the size of \( \text{PERF}(R_i) \) and \( \min(m\_set, nm\_set) \), send the smaller one back to \( R_i \) to reduce \( R_j \). The transmission cost is the size of the smaller one which has been sent. If the semijoin between relation \( R_i \), \( R_j \) over join attribute \( A \) does not exist in the executed semijoin list which means there was no semijoin happened from \( R_i \) to \( R_j \) on attribute \( A \). so we need to consider to do the semijoin or not by the term of net benefit >0. If net benefit > 0, we do this semijoin, the cost is \( |R_j[A]| \times \text{width of } A. \) otherwise we leave 2 relations without executing semijoin.

8. Repeat steps 4,5,6,7 till all of schedules have been processed.

9. Output the transmission cost which is the sum of every reduced relation size plus the transmission cost in every single schedules.

3.2.3 Example of Proposed Algorithms

The following example will show how the algorithm Perf join and Bloom Filter join work. Suppose we have a query “List the P#, PNAME and total quantity for all parts that are current on order from suppliers who supply that part to jobs.” In this query, there are two joining attribute which are P# and S#. Assume that each relation is located at different network node. After the local processing, the query can be represented as the SQL listed below:
"select * from PARTS, ORDER, SPJ where PARTS.P#=ON-ORDER.P#=SPJ.P#
and ON-ORDER.S#=SPJ.S#."
Step4: Order the schedules by ascending size of join attribute projection of the first relation of each schedule.

\[
\begin{align*}
\text{S}: & \quad R_1 \rightarrow R_2 \rightarrow R_1 & R_1[S#] = 100 \\
\text{P}: & \quad R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow R_2 \rightarrow R_1 & R_1[P#] = 400
\end{align*}
\]

Step5: Start to execute the schedule for S: R_1 \rightarrow R_2 \rightarrow R_1.

1) Execute the forward reduction phase R_1 \rightarrow R_2:

We need to decide the bloom filter join form R_1 to R_2 over attribute S# should be executed or not. The BF(R_1[S#]) = domain size/8 = 500/8 = 63 bytes which is smaller than the projection size of S#. The benefit is 1600 which derived by size of R_2*(1- p (R_1[S#])). The Net Benefit = Benefit - Cost=1600-63>0, so we need to do this bloom filter join. After execute this bloom filter join, Relation R_2 has been reduced to R_2' with the size 400 derived from 2000 * 0.2. The attribute size and selectivity of joining attribute has been updated to 90, and 0.18 respectively. We record this bloom filter join in the bloom filter join list.

2) Execute the backward reduction phase R_2 \rightarrow R_1:

First we search the bloom filter join list to see the bloom filter join between R_1 and R_2 over the attribute S# exists or not. In this case, it is in, which means that the bloom filter join from R_1 to R_2 has been done. We can generate bloom filter for the match set, which is also 500/8=63. For here, because the size of match set m_set= 20(100*0.2) is smaller, Then we just need to send the match set back to the relation R_1. And the relation R_1 will be reduced to R_1' with the size 180.

After that, there is no more relation to be considered in the schedule S#. The transmission cost of schedule S# is 63+20 bytes, the size of relation R_1 and R_2 has been reduced to 180 and 400 respectively. The execution schedule for S# has been done.
We start to process next schedule for P# with the same process with schedule S#. Our algorithm keeps running until there are no more schedules left. After that, all reduced the relations will be sent to the final query site to participate the global query. The output of our algorithm is sum of all reduced the relations' size plus each sum of the cost occurred in the every single schedule.

**Example of Algorithm PERF:**

For the perf join, the process from step1 to step 4 will be same as the Algorithm BF. It will generate same execution schedules for the query listed below:

\[
\begin{align*}
S#: & \quad R_1 \rightarrow R_2 \rightarrow R_1 \\
P#: & \quad R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow R_2 \rightarrow R_1
\end{align*}
\]

**Step5:** Start to execute the schedule for S#: \( R_1 \rightarrow R_2 \rightarrow R_1 \).

1) Execute the forward reduction phase \( R_1 \rightarrow R_2 \):

We need to decide the semijoin form \( R_1 \) to \( R_2 \) over attribute S# should be executed or not. The cost is 100. Benefit is 1600 which derived by size of \( R_2 \)\((1- \rho (R_1[S#])) \). The Net Benefit = Benefit – Cost=1600-100=1500>0, so we need to do this semijoin. After execute this semijoin, Relation \( R_2 \) has been reduced to \( R_2' \). The size of \( R_2' \) is 400(2000 * 0.2) now. The attribute size and selectivity of joining attribute has been updated to 90, 0.18 respectively. We record this semijoin in the semijoin list.

2) Execute the backward reduction phase \( R_2 \rightarrow R_1 \):

First we search the semijoin list to see the semijoin between \( R_1 \) and \( R_2 \) over the attribute S# exists or not. In this case, it is in, which means that the semijoin from \( R_1 \) to \( R_2 \) has been done and site \( R_2 \) already has the join attribute projection information of \( R_1[S#] \). It can generate PERF (\( R_1[S#] \)) which size is \(|d|/8 = 100/8=13\) bytes. The size of match set \( m\_set=20 \) (100*0.2), and the size of none match set \( nm\_set = 80 \). Comparing the size between PERF (\( R_1[S#] \)) and \( \min (m\_set, nm\_set) \), we find the smaller one is the PERF (\( R_1 \)). Then we can do PERF join on back
reduction phase. The cost is the size of PERF which is 13 and the relation R_1 will be reduced to R_1' with the size 180.

After that, there is no more relation to be considered in the schedule S#. The cost of schedule S# is 100+13 bytes, the size of relation R_1 and R_2 has been reduced to 180 and 400 respectively.

We start to process next schedule for P# with the same process with schedule S#. Our algorithm keeps running until there are no more schedules left. After that, all reduced the relations will be sent to the final query site to participate the global query. The output of our algorithm is sum of all reduced the relations' size plus each sum of the cost occurred in the every single schedule.
Chapter 4 Experiment and Evaluation

To evaluate the actual performance of the proposed algorithms, we carried out multitudinous experiments based on various scenarios and queries. In this chapter, we describe our methodology, present detail of our experiments, and discuss the final result.

4.1 Methodology

The platform for evaluating the proposed algorithms has to achieve the following objectives.

1. To measure the performance enhancement of Algorithm PERF and Algorithm BF over the IFS respectively.
2. To compare the performance of Algorithm PERF and Algorithm BF under a wide variety of distributed queries.

The following formulas are used to calculate the performance enhancement between the algorithms:

**IFS vs. Proposed algorithm:**
\[
\frac{\text{Cost}(\text{IFS}) - \text{Cost}(\text{Proposed Algorithm})}{\text{Cost}(\text{IFS})} \times 100\% = \text{Percentage Improved}
\]

**Algorithm 1 vs. Algorithm 2:**
\[
\frac{\text{Cost}(\text{Algorithm 1}) - \text{Cost}(\text{Algorithm 2})}{\text{Cost}(\text{Algorithm 1})} \times 100\% = \text{Percentage Improved}
\]
From experiment, we need to find out the answers for the following involved questions.

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

4.2 Test Query and Platform

The experiment system involves query generator, proposed algorithms and the analysis program. Some details will be given later in this section.

4.2.1 Test Query (query generator)

The experiments system takes large amount of queries as input to evaluate proposed algorithms. The form of a query we have already represented in Figure 8. It contains the following characteristics:

- Number of relations: Each query consists of arbitrary number of relations from 3-6.

- Number of join attributes: Each relation have arbitrary number of join attributes from 2-4.

- Relation cardinality: the number of tuples or records in a relation. Each relation in the query has between 500 and 4000 tuples.

- Attributes domain size: the total number of distinct attribute values an attributed can possible contain. We fix all join attribute domain size to 1000 in our experiment system.

- Selectivity: the ratio of distinct attribute values out of the number of all possible values of a join attribute. Suppose the cardinality of the joining
attribute is |d|, the domain of the d is Dom, p=|d|/Dom (d). Generally, the selectivity of an attribute is an estimate of the ability of the attribute to reduce the size of the relations. A joining attribute has high selectivity if the ratio is low while low ratio denotes the high selectivity. For example, a selectivity of 0.1 is considered high while a selectivity of 0.9 is low.

- Query Type: In our experiment platform, the relations is picked from 3 to 6, the arbitrary number of joining attributes is form 2-4. Selectivity is divided to 3 level which are high (0.1-0.4), med (0.4-0.7), low (0.7-0.9). We use these 3 factors to name a query type. A type of 6-3-0 represents a query which has 6 relations, 3 joining attributes and the selectivity range of the attribute is from 0.1 to 0.4. With this rule, we will have 36 different kind of queries total. For each kind of query type, we will generate 10 queries with random data, and get the average output as final result.

We save these kind of information as XML format file. The example fragment of XML file listed below in Figure 8 denotes one query. 360 pieces of fragment like this one constructs the input queries file of our propose algorithms.
4.2.2 Platform Implementation

The experiment platform is developed with Microsoft C# based on object-oriented concept. Several classes as show in the following figure are constructed in this program to represent the data structures used in the algorithm. The hierarchy of basic class tree is illustrated in Figure 9.
The descriptions of above classes are as follows:

- **Query Maker**: for each kind of query type (e.g., 6-4-0), generate 10 queries' xml nodes. Save all these nodes to one xml file. The detail rule of this process has been introduced in 4.2.1.

- **Query**: For each query node in the read in xml file, we generate a query. Each query has an arraylist to save the schedules of the query. The process of the query generating is also the process of initialization for schedule, relation, and attribute.

- **Schedule**: Represent the sequence of the relation execution. Each schedule instance contains a list of relation ordered by the size of the
projection of the attribute it refers to. It also has an array list to mark the semijoin which has been executed by which 2 relations. The cost used to record how many bytes has been transferred during this schedule.

- **Relation:** Represent the relation's information like name, size and contained attributes. Attributes list use to keep the joining attributes information of this relation.

- **Attribute:** Represent single joining attribute information in a relation such as cardinality, name, selectivity and size.

- **Sorter:** use to sort the size or cost etc....

- **Semijoin:** represent a semijoin was executed by which 2 relations over the joining attributed. In the backward reduction phase, we need to decide whether we should apply the proposed join to the schedule by the term of checking a semijoin has been done between these 2 relation during the forward reduction phase or not.

- **Algorithms:** take a query as input, run it with algorithm IFS, HASH, PERF respectively, and output is the total cost for each of algorithm. The total cost is the summation of the transmission total cost and total reduced relation size.

- **Database:** once an instance of a single query has been done by the algorithms. We will insert the query result data into the database.

- **Result:** use to analysis and display the experiment data.
4.3 Result Evaluation

After a 10 queries for 36 types of queries as input to our proposed algorithms, we have the experiment result data listed below:

<table>
<thead>
<tr>
<th>Type</th>
<th>IFS</th>
<th>BF</th>
<th>IMP(BF)</th>
<th>PERF</th>
<th>IMP(PERF)</th>
<th>BF/PERF</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-2-0</td>
<td>3854</td>
<td>758</td>
<td>80.33%</td>
<td>1100</td>
<td>71.46%</td>
<td>45.12%</td>
</tr>
<tr>
<td>3-2-1</td>
<td>7388</td>
<td>2190</td>
<td>70.36%</td>
<td>3864</td>
<td>47.70%</td>
<td>76.44%</td>
</tr>
<tr>
<td>3-2-2</td>
<td>7152</td>
<td>4838</td>
<td>32.35%</td>
<td>6056</td>
<td>15.32%</td>
<td>25.18%</td>
</tr>
<tr>
<td>3-3-0</td>
<td>5848</td>
<td>834</td>
<td>85.74%</td>
<td>1110</td>
<td>81.02%</td>
<td>33.09%</td>
</tr>
<tr>
<td>3-3-1</td>
<td>8630</td>
<td>2518</td>
<td>70.82%</td>
<td>4380</td>
<td>49.25%</td>
<td>73.95%</td>
</tr>
<tr>
<td>3-3-2</td>
<td>12696</td>
<td>5762</td>
<td>54.62%</td>
<td>10162</td>
<td>19.96%</td>
<td>76.36%</td>
</tr>
<tr>
<td>3-4-0</td>
<td>9588</td>
<td>1132</td>
<td>88.19%</td>
<td>1550</td>
<td>83.83%</td>
<td>36.93%</td>
</tr>
<tr>
<td>3-4-1</td>
<td>15052</td>
<td>2800</td>
<td>81.40%</td>
<td>5260</td>
<td>65.05%</td>
<td>87.86%</td>
</tr>
<tr>
<td>3-4-2</td>
<td>12494</td>
<td>5446</td>
<td>56.41%</td>
<td>7730</td>
<td>38.13%</td>
<td>41.94%</td>
</tr>
<tr>
<td>4-2-0</td>
<td>7610</td>
<td>826</td>
<td>89.15%</td>
<td>1176</td>
<td>84.55%</td>
<td>42.37%</td>
</tr>
<tr>
<td>4-2-1</td>
<td>9392</td>
<td>2240</td>
<td>76.15%</td>
<td>3748</td>
<td>60.09%</td>
<td>67.32%</td>
</tr>
<tr>
<td>4-2-2</td>
<td>10124</td>
<td>4208</td>
<td>58.44%</td>
<td>6966</td>
<td>31.19%</td>
<td>65.54%</td>
</tr>
<tr>
<td>4-3-0</td>
<td>10776</td>
<td>1134</td>
<td>89.48%</td>
<td>1544</td>
<td>85.67%</td>
<td>36.16%</td>
</tr>
<tr>
<td>4-3-1</td>
<td>13910</td>
<td>2820</td>
<td>79.73%</td>
<td>5242</td>
<td>62.31%</td>
<td>85.89%</td>
</tr>
<tr>
<td>4-3-2</td>
<td>13874</td>
<td>5434</td>
<td>60.83%</td>
<td>8190</td>
<td>40.97%</td>
<td>50.72%</td>
</tr>
<tr>
<td>4-4-0</td>
<td>15582</td>
<td>1556</td>
<td>90.01%</td>
<td>2140</td>
<td>86.27%</td>
<td>37.53%</td>
</tr>
<tr>
<td>4-4-1</td>
<td>14510</td>
<td>2880</td>
<td>80.15%</td>
<td>5614</td>
<td>61.31%</td>
<td>94.93%</td>
</tr>
<tr>
<td>4-4-2</td>
<td>18976</td>
<td>5888</td>
<td>68.97%</td>
<td>14692</td>
<td>22.58%</td>
<td>149.52%</td>
</tr>
<tr>
<td>5-2-0</td>
<td>11120</td>
<td>1060</td>
<td>90.47%</td>
<td>1448</td>
<td>86.98%</td>
<td>36.60%</td>
</tr>
<tr>
<td>5-2-1</td>
<td>11934</td>
<td>2528</td>
<td>78.82%</td>
<td>4174</td>
<td>65.02%</td>
<td>65.11%</td>
</tr>
<tr>
<td>5-2-2</td>
<td>13766</td>
<td>4356</td>
<td>68.36%</td>
<td>9666</td>
<td>29.78%</td>
<td>121.90%</td>
</tr>
<tr>
<td>5-3-0</td>
<td>13414</td>
<td>1426</td>
<td>89.37%</td>
<td>1986</td>
<td>85.19%</td>
<td>39.27%</td>
</tr>
<tr>
<td>5-3-1</td>
<td>14960</td>
<td>2792</td>
<td>81.34%</td>
<td>5066</td>
<td>66.14%</td>
<td>81.45%</td>
</tr>
<tr>
<td>5-3-2</td>
<td>17238</td>
<td>4708</td>
<td>72.69%</td>
<td>10672</td>
<td>38.09%</td>
<td>126.68%</td>
</tr>
<tr>
<td>5-4-0</td>
<td>14876</td>
<td>1502</td>
<td>89.90%</td>
<td>2014</td>
<td>86.46%</td>
<td>34.09%</td>
</tr>
<tr>
<td>5-4-1</td>
<td>16694</td>
<td>2876</td>
<td>82.77%</td>
<td>4870</td>
<td>70.83%</td>
<td>69.33%</td>
</tr>
<tr>
<td>5-4-2</td>
<td>24092</td>
<td>5878</td>
<td>75.60%</td>
<td>13456</td>
<td>44.15%</td>
<td>128.92%</td>
</tr>
<tr>
<td>6-2-0</td>
<td>11226</td>
<td>918</td>
<td>91.82%</td>
<td>1160</td>
<td>89.67%</td>
<td>26.36%</td>
</tr>
<tr>
<td>6-2-1</td>
<td>14572</td>
<td>2516</td>
<td>82.73%</td>
<td>4106</td>
<td>71.82%</td>
<td>63.20%</td>
</tr>
<tr>
<td>6-2-2</td>
<td>17206</td>
<td>4826</td>
<td>71.95%</td>
<td>11764</td>
<td>31.63%</td>
<td>143.76%</td>
</tr>
<tr>
<td>6-3-0</td>
<td>13182</td>
<td>1094</td>
<td>91.70%</td>
<td>1430</td>
<td>89.15%</td>
<td>30.71%</td>
</tr>
<tr>
<td>6-3-1</td>
<td>20586</td>
<td>3564</td>
<td>82.69%</td>
<td>5820</td>
<td>71.73%</td>
<td>63.30%</td>
</tr>
<tr>
<td>6-3-2</td>
<td>21894</td>
<td>5304</td>
<td>75.77%</td>
<td>14544</td>
<td>33.57%</td>
<td>174.21%</td>
</tr>
<tr>
<td>6-4-0</td>
<td>16192</td>
<td>1630</td>
<td>89.93%</td>
<td>2198</td>
<td>86.43%</td>
<td>34.85%</td>
</tr>
<tr>
<td>6-4-1</td>
<td>27624</td>
<td>4708</td>
<td>82.96%</td>
<td>8458</td>
<td>69.38%</td>
<td>79.65%</td>
</tr>
<tr>
<td>6-4-2</td>
<td>26086</td>
<td>5974</td>
<td>77.10%</td>
<td>14806</td>
<td>43.24%</td>
<td>147.84%</td>
</tr>
</tbody>
</table>

**Figure 10: Experiment Result of Algorithm IFS, BF and PERF**
Comparison of Transmission Cost

Figure 11: Transmission Cost Comparison of IFS, BF and PERF
Figure 12: Reduction Rate Comparison between BF and PERF

Figure 11 and 12 show the transmission cost comparison of BF, PERF with IFS and transmission reduction rate comparison of propose algorithms over the IFS. Our propose algorithms make greater improvement on performance than IFS. Algorithm BF exceeds the IFS over 91.82% at most while Algorithm PERF can make at most 89.67% improvement.
Figure 13 shows the improvement percentage of Algorithm BF over Algorithm PERF derived by the formula listed in 4.1. As it is showed in the above chart, the peak values represent the different between 2 algorithms. The larger value the query type is, the more transmission cost can be reduced than the algorithm PERF produced. We also noticed that with more join attributes and relations join into the query, the different of performance between two algorithms produced almost keep the same level in the low selectivity range. But the algorithm BF produces perceptible enhanced performance with a higher selectivity.

Conclusion can be driven from the above figures that comparing to IFS, both algorithm BF and PERF are able to produce a high reduction power over distributed query transmission cost. The average results for both algorithm shows that the BF is over-performed than PERF. However, to find out and answer for the questions we listed in 4.1. We need to do further investigation on other factors affecting query transmission, such as selectivity, number of relation and join attributes, domain size etc…
4.3.1 Effects of the Selectivity Level

In this scenario, we will find how the cost reduction rate of the proposed algorithms will be related by selectivity. As we know, selectivity is defined as a ratio of distinct attribute values over the attribute domain size. As the most important factor in a distributed query, it can be use to estimate the reduced size of a join attribute during a semijoin operation. The selectivity range in the query is between 0.1 and 0.9.

Figure 14 illustrates reduction rate comparison with the low selectivity level for both algorithms. BF improves the reduction rate from a low percentage of 80.3% to 91.82%, while Algorithm PERF does from 71.5% to 89.7%.

Experimental data shows that, both Algorithm BF and Algorithm PERF show the great performance in high selectivity level. The more relations and join attribute a query has, the better reduction rate that both algorithms produce. However, with the distributed query of high selectivity, the performance of Algorithm BF outperforms than Algorithm PERF.
As it shows in Figure 15, with medium selectivity, the lowest reduction rate that the proposed algorithms can produce is at least 67.7% by Algorithm BF as it is 58.5% by Algorithm PERF. The highest value of improved reduction rate from BF is 85.7% as it is 86.9% by PERF. The reduction rate begins to descend as selectivity grows.

At a selectivity range of 0.4-0.7, the performance both algorithms produced is still competitive and keeping growing with an increasing relation and attribute number. Rather remarkable, BF over-performed Perf at the beginning but failed to keep the advanced position later on after relation and attribute reach certain numbers.
As it shows in Figure 16, with low selectivity, the lowest reduction rate that the proposed algorithms can produce is at least 32.4% by Algorithm BF as it is 15.3% by Algorithm PERF. The highest value of improved reduction rate from BF is 77.1% as it is 44.2% by PERF. Reduction rate reaches its lowest value while selectivity decreases its own range.

When the selectivity is very low, both Algorithm BF and Algorithm PERF’s performance decrease obviously. Especially for PERF, the reduction rate is down to around 15% under low relation and attribute. Yet, performance of both algorithms bounced back with the growth of relations and attributes.
4.3.2 Effects of the Number of Relations

In this section, we will find how the cost reduction rate of the proposed algorithms will be related by the number of the relations.

![Reduction Rate with Different Relations](image)

Figure 17: Reduction Rate with Different Relation

In figure 17, with the queries of relation range from 3 to 6, the average of reduction cost that bloom filter join can produce better than perf does. However, with more relation joining to the query, the performance of both algorithms is getting better.
4.3.3 Effects of the Number of Attributes

In this section, we will examine how the number of join attributes will affect the cost reduction rate of the proposed algorithms.

In figure 18, from 2 attribute to 4 attributes the average of reduction cost of bloom filter join still better than perf. However, under the condition of involving more join attributes, a more competitive reduce rate will be demonstrated.
4.3.4 Effects of the Domain Size

In this scenario, we are looking forward to present a methodology on measuring that how the domain size will influence the performance of our proposed algorithms, in a more self-revealing way. We pick up a group of queries with 3 relations, 2 join attributes and medium selectivity level (3-2-1) and try to increase their domain size from 1000 to 2000. Due to the fact that domain size is a responsive parameter of our algorithms – it produces a direct effect on bloom filter size. Since domain variable will cause the change to cardinality, and therefore shape PERF size accordingly.

![Reduction Rate with Different Domain Size](image)

**Figure 19: Reduction Rate with Different Domain Size**

As it is illustrated in the experiment, from the Figure 19. We notice that, both of our algorithms performance seems to increase slightly with the growth of domain size.
4.4 Evaluation and Discussion

In our experiment platform, after large amount of queries with arbitrary number of relations from 3 to 6, join attributes from 2 to 4, selectivity range from low to high were executed as input for our propose algorithms. The output result data show that both of Algorithm PERF and Algorithm BF make greater enhancement over the Initial Feasible Solution (IFS). By comparing with between PERF join and Bloom Filter join, we have following conclusion.

- The reduction rate of proposed algorithms increase by the selectivity level increase. In another words, with same numbers of relations and attributes, when the selectivity in the low level, our proposed algorithms have the best performance.
- The number of relations is an important factor for both of proposed algorithms. More relations the distributed query has, better reduction power it would produce.
- The number of join attributes is another important factor to affect the query performance. The query has more attributes participated will have better performance.
- The domain size doesn't affect our proposed algorithms too much. With the increase of domain size, the distributed under the same condition of selectivity, relations and join attribute, the reduction rate for both algorithms have slightly improvement.

Both our Algorithms perform better in the high level selectivity condition, while getting worse with the selectivity level getting lower. In general speaking, Algorithm BF performs better in almost all circumstance. Especially with a lower selectivity, when algorithm PERF declines its performance to a great extent. Algorithm BF can still produce a rather effective reduction rate.

A traditionally more acceptable understanding was that Perf join could perform better than bloom filter, because it has similar data storage and never encounter the information loss due to the hash function. However, in our experimental
environment, we assume that a perfect hash function has been applied which means no false drop would happen. Plus, we applied bloom filter twice on both forward reduction phase and backward reduction phase in two-way semijoin. Especially in forward reduction phase. The bloom filter based algorithm can effectively reduce the transmission cost while perf based 2-way join algorithm which still send original semijoin projection. Thus it is rather safe to conclude that two-way bloom filter based algorithm performs better than perf join.
Chapter 5 Conclusion and Future work

5.1 Conclusion

The main purpose of distributed query processing is to find the best sequence of database operation to minimize the transmission cost. Because most of implemented algorithms in this area are heuristic, our objective is to figure out the near-optimal solutions.

The main approaches in distributed query processing can be classify to 3 categories, which are join based, semijoin based and filter based. Compare with the join based algorithms always involve the large data transmission, semijoin acts as a powerful reducer in distributed query processing which only send the join attribute projection to instead of delivering the whole relation from one relation to another. As a result, the tuples which are not contributive for the query will be eliminated before they sent to final query site. Semijoin based algorithm still spend a lot even applied it’s extend version 2-way semijoin by giving extra backward reduction. Filter based algorithms proposed as cheap prototype which use a bit vector to represent the semijoin projection information during the data transmission. Since bloom filter implement with hash function, the false drop caused by hash collusion can not be avoid. Perf was brought into this field as a novel solution. It use a small size bit vector (perf) based on tuples scan order to encode the join information during backward reduction phase of 2-way join. Compare with bloom filter join, it has same data storage with bloom filter and would never involve the information loss caused by hash collusion.

From above point of view, we know that the cheapest solution is to represent the original semijoin projection with a smaller size bit vector during the data transmission. In this thesis, we propose two filter-based algorithms by applying PERF and Bloom Filter technique on 2-way semijoin algorithm respectively. In algorithm PERF, we construct the perf base on tuple scan order in backward
reduction phase while with algorithm, we apply the bloom filter for semijoin projection in forward reduction phase while generate bloom filter for join match set in backward way.

After run large amount experiments with distributed queries as input. Both of algorithm PERF and algorithm BF make show their reduction power by comparing with Initial Feasible Solution. The evaluation between 2 proposed algorithms show that algorithm BF outperform algorithm PERF in most condition especially with a lower selectivity. In this kind of situation, while algorithm PERF declines its performance to a great extent. Algorithm BF can still produce a rather effective reduction rate.

Perf should outperform BF if we just compare them simply. However, in our experimental environment, we adopt perfect hash function to address tuple. Plus, we applied bloom filter twice on both forward reduction phase and backward reduction phase base on two-way semijoin. Due to the intent of perf is to reduce the transmission cost during the backward reduction. There is still expensive cost in forward reduction phase. With algorithm BF we proposed, it eliminates large transmission cost in forward phase. Even think about the false drop in tolerable range, algorithm BF can still show its good performance.

5.2 Future Work

In our experiment platform, we performed 10 queries for each kind of query over 36 query types. However, in order to get more accurate and persuasive result from experiments, we need to endeavor larger amount of query for the proposed algorithm. We also need to increase the size for the relations and domain to make the result appropriate for real-life circumstances and applications.
In Algorithm BF, to simplify the question, we assume that we use the perfect hash function which means no false drop would happen. While in the real case, collision can not be avoided in the hash filter based algorithm. So there are still some spaces to continue our research under the situation with collision and try to find optimal hash function to apply on the semijoin projection to minimize the transmission cost.

In Algorithm PERF with very high or very low semijoin selectivity, it will contain large amount of 0 or 1. We can try to compress the encode join information by sending the address of 1 or 0 but within Perf [LR95] to gain extra reduction during the transmission.
Appendix: Testing Environment

Hardware:
- Dell Latitude D620
- Base: Intel® Core(TM) 2 CPU T5600 @1.83 GHz
- Memory: 1GB Dual Channel DDR2 SDRAM at 400MHz (4x256M)

Software:
- Windows XP Professional SP2
- Microsoft Visual Studio 2003
- Microsoft .Net Framework 1.1
Bibliography


[PC90] W. Perrizo and C. Chen “Composite Semijoins in Distributed Query Processing” Information Science, March. 1990

[Q88] G. Z. Qadah, Filter-based join algorithm on uniprocessor and
distributed memory multiprocessor database machines, vol. 303
of Lecture Notes in Computer Science, pp. 388-413. Springer-

[RK91] N. Roussopoulos and H. Kang "A Pipeline N-way Join Algorithm
Based on the 2-way Semijoin Program" IEEE Trans, on
486-495

[TC92] J. Tseng and A. P. Chen "Improving Distributed Query
Processing by Hash-Semijoins" Journal of Information Science
and Engineering, 1992

processing in a distributed data base. In Proceedings of the 1983
ACM SIGSMALL symposium on Personal and small computers,
pages 237–244, 1983

[V02] Patrick Valduriez “Principles of Distributed Database Systems”

[VG84] P. Valduriez and G. Gardarin, “Join and semijoin algorithms for a
multiprocessor database machine,” ACM Transactions on


[WLC91] C. Wang, V. Li, and A. Chen, “Distributed Query Optimization by
One-Shot Fixed-Precision Semijoin Execution” in Proc. 7th Znt.
Con$ on Data Engineering, pp. 756-763, 1991

Based Algorithm”. Master thesis, University of Windsor, 2005

processing strategy in a distributed database environment” In
SIGMOD 83, Proceedings of the Annual Meeting (San Jose,

Vita Auctoris

Name: Ming Pei

PLACE OF BIRTH: Wuhan, Hubei, China

YEAR OF BIRTH: 1976

EDUCTION: M.Sc., Computer Science,
University of Windsor,
Windsor, Ontario, Canada
2004 ~ 2007

B.Sc., Computer Science,
Navy University of Engineering,
Wuhan, Hubei, China
1994 ~ 1998