DISTINCT ENCODED RECORDS JOIN OPERATOR FOR DISTRIBUTED QUERY PROCESSING

A Thesis submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in Computer Engineering

by
Ahmet Cumhur ÖZTÜRK

December 2012
İZMİR

We approve the thesis of **Ahmet Cumhur ÖZTÜRK**

Examining Committee Members:

_____

Assist. Prof. Dr. Belgin ERGENÇ
Department of Computer Engineering,
İzmir Institute of Technology

_____

Assist. Prof. Dr. Tolga AYAV
Department of Computer Engineering,
İzmir Institute of Technology

_____

Assist.Prof. Dr. Asil ALKAYA
Department of Management Information Systems,
Adnan Menderes University

14 December 2012

_____

Assist. Prof. Dr. Belgin ERGENÇ
Supervisor, Department of
Computer Engineering
İzmir Institute of Technology

_____

Prof. Dr. Sıtkı AYTAÇ
Head of the Department
of Computer Engineering

_____

Prof. Dr. R. Tuğrul SENGER
Dean of the Graduate School of
Engineering and Sciences

ACKNOWLEDGEMENTS

This thesis work could not have been accomplished without the support of my advisor. I would like to express my appreciation to Asst.Prof.Dr.Belgin ERGENÇ whose tremendous support, guidance, stimulating ideas and review of this thesis report were invaluable to successful completion of this thesis. Without her guidance and patience, I might never have developed the great interest in Computer Science that I have today.

ABSTRACT

DISTINCT ENCODED RECORDS JOIN OPERATOR FOR DISTRIBUTED QUERY PROCESSING

Nowadays distributing data among different locations is very popular due to needs of business environment. In today's business environment, accessible, reliable, and scalable data is a critical need and distributed database system provides those advantages. It is a need to transfer data between sites while processing query in distributed database system, if the connection speed between sites is low then transmitting data is very time consuming. Optimizing distributed query processing is different from optimizing query processing in local database system. Most of the algorithms generated for distributed query processing focus on reducing the amount of data transferred between sites.

Join operation in database system is for combining different tables with a common join attribute value, if the tables that are put in a join operation are at different locations then some of the tables are needed to be transferred to between sites. Join operation optimization algorithms in distributed database system focus on reducing the amount of data transfer by eliminating redundant tuples from relation before transmitting it to the other site.

This thesis introduces a new distributed query processing technique named distinct encoded records join operation (DERjoin) which considers duplicated join attributes in a relation and eliminates them before sending the relation to another site.

ÖZET

DAĞITIK SORGU İŞLEME İÇİN AYRI KODLANMIŞ KAYITLAR BİRLEŞTİRME İŞLETMENİ

Günümüzde iş ortam ihtiyaçlarından dolayı veriyi farklı konumlara dağıtmak çok popülerdir. Bugünün iş ortamında erişilebilir, güvenilir ve ölçeklen dirilebilir veri kritik bir ihtiyaçtır ve dağıtık veri tabanı sistemi bu avantajları sağlar. Dağıtık veri tabanı sisteminde sorgu işlenirken genelde konumlar arası veri transferi gereklidir, eğer konumlar arası bağlantı hızı düşük ise veri transferi en çok zaman alan iş olabilir. Dağıtık sorgu işlemeyi optimize etmek yerel veritabanında sorgu işlemenin optimize edilmesinden farklıdır, geliştirilen çoğu dağıtık sorgu işleme algoritması konumlar arası transfer edilen veriyi azaltmaya odaklanır.

Veri tabanında birleştirme işlemi ortak bir birleştirme değeri ile farklı tabloların birleştirilmesidir, eğer birleştirme işlemi yapılan tablolar farklı konumlarda ise bazılarının diğer konumlara transfer edilmesi gerekir. Dağıtık veri tabanı sisteminde birleştirme işlemi optimizasyon algoritmaları, tabloları diğer konuma aktarmadan önce içerdikleri gereksiz verileri tablodan çıkartarak, transfer edilen veri miktarını azaltmaya odaklanır.

Bu tez farklı kodlanmış kayıtlar birleştirme işlemi(DERjoin) adında, tablo içerisindeki çift birleştirme değerlerini göz önüne alan ve tabloyu diğer bir konuma göndermeden önce bunları tablodan çıkaran yeni bir dağıtık sorgu işleme tekniği sunmaktadır.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Distributed database system consists of physically separated databases connected with a computer network. A database is a collection of related data, by data it is mean known facts that can be recorded and that have implicit meaning [23]. Distributing data to computers is both feasible and needed, it is feasible because of the recent technological advances and it is needed because business requirements have been changing, which made distributing data cost effective, businesses are beginning to rely on distributing data rather than centralizing the data in one database [2], it is more advantageous than local databases and some of these advantages might be as follows [23]:

Increased reliability and availability: If the data is distributed over several sites and one of the sites fail then other sites is able to operate, only the data at the failing site cannot be accessed.

Improved performance: It is possible to break up a query into a number of sub queries that execute in parallel, so execution of the query can be separated through sites and reduces CPU and I/O usage at each site.

Easier to expand: Adding more data, increasing database sizes or adding more processors is much easier.

It should be possible to update, manage, delete or access the data stored in a database, and database management system is a software that is developed for that purposes. A database management system is a collection of programs that enables users to create and maintain database [1]. A user can query data by using database management system.

Join operation is the most fundamental and the most difficult relational query operation to implement efficiently [19]. The join operation combines related tuples from the two different relations. It often needs to transfer data between sites to combine the relations located at different sites. Cost of transmitting data between sites shadows local

processing cost if the sites are not connected with high speed network and most distributed database management system query optimization algorithms consider reducing the amount of data transfer for optimization criterion. Query optimization process estimates the cost of various queries processing plans and chooses the most cost effective plan for processing the query. Query processing in a distributed system can also be accelerated by using parallel processing. Increasing the computer number to process the query reduces the elapsed time for processing the query with another meaning it is going to reduce total response time of the query. The advantage of parallel processing relies on hardware devices, and also parallel processing ignores the amount of data transferred between sites while processing a query. However in a low bandwidth connected distributed system, communication cost is the most effective cost factor while processing a query. Most of the distributed join operation algorithms based on reducing the amount of data transfers process by the following three phases [24]:

1. **Local processing phase:** all local processing that requires no intersite communication is performed at each site involved in the query.
2. **Reduction phase:** the size of relations or/and intermediate results are reduced by using semijoin to reduce the transmission cost.
3. **Final query processing phase:** reduced relations and/or intermediate results are sent to the final processing site.

Most of the distributed query processing algorithms are focused on the second step in which the reduction of data is performed in order to reduce the total amount of data transfer between sites.

Semijoin [18], is one of the most popular join approach in distributed query processing for reducing the transmission cost, and there are many previous join algorithms based on semijoin operation. Semijoin and semijoin based operations reduce the amount of data transfer by eliminating the redundant tuples from the relation before sending data to the other site, to eliminate the redundant tuples a tiny piece of data need to be exchanged between sites and this data is the projection of the join attributes of one relation. A relation may consist of many duplicated join attributes. If those duplicated join attributes are not eliminated before sending them to the other site, significant amount of redundant data can be transferred as stated in [1].

This thesis presents a novel semijoin based distributed join operation algorithm called Distinct Encoded Records (DERjoin) operation. In DERjoin operation reducing the amount of data transferred between sites by using semijoin operation and

compressing the records before transmitting them is considered. For example, if join attributes of relations are long strings then they may be dozen of bytes, to reduce the amount of data transferred between sites only eliminating redundant tuples will not be the best solution to reduce the total query processing time, but compressing the records before transmitting them should also be considered.

In performance evaluation, DERjoin operation is compared with Positionally Encoded Record Filters (PERFjoin) operation. PERFjoin operation is chosen because it compresses the data before transmission and also it does not use any hash function to compress the data as in DERjoin operation.

The thesis is organized as follows: Chapter 2 presents a thorough discussion of the distributed system and distributed query processing. Chapter 3 presents discussion of the studies in the related literature. Chapter 4 presents the Distinct Encoded Records join (DERjoin) operation. Chapter 5 illustrates processing a distributed join operation by using DERjoin and PERFjoin operation and performance comparison with DERjoin and PERFjoin operations. Chapter 6 is the conclusion chapter and includes a brief summary of the study and suggestions for further research.

CHAPTER 2

DISTRIBUTED QUERY PROCESSING

In this section first the distributed database system and textbook architecture of distributed query processing are introduced. Then the cost measures in distributed query processing and two popular query optimization strategies SDD1 and Algorithm General are explained.

2.1. Distributed Database System

Distributed Database System is a collection of multiple, logically interrelated databases distributed over a computer network, and distributed database management system is the software that permits the management of distributed database and makes the distribution transparent to the user[6]. The data in the distributed database is distributed over different sites, and each site in the network has the ability of independent processing to perform local application [8]. While local database is a collection of data which can be accessed locally, distributed database consists of a collection computers either located in the same location or located at different sites. Using distributed database system is a common thing for organizations that have branches geographically located in different places. The distribution of data offers some advantages over the centralization of data at a single computer and these advantages include [1,6];

- Increased data reliability; the same data can be stored at more than one site. So, if a site crashes or communication link fails then the same data can be find in another site through another communication link without any data lost.
- Potential upwards scaling of data capacity; it is possible to increase database size by adding extra site.
- Data can be shared among sites.

On the other hand, there are some difficulties in building and managing distributed database as stated in [1, 6];

- Distributed database systems are more complex than centralized databases. So, data communication, concurrence and synchronization of operations should be considered.
- Distributed database systems are more expensive than the centralized systems because both of the hardware and software of the distributed database are more complex than centralized systems.
- If some of the sites or communication link between some of the sites fail while an update is being executed the system must guarantee that update is reflected on the data located at the failing or unreachable site.

2.2. Phases of Distributed Query Processing

Textbook architecture of query processing is presented by [2] and it is shown in Figure 2.1. This architecture is designed for Starburst project and also it can be used for any kind of database system including distributed, centralized or parallel databases. Also it is emphasized in [2] that Starburst architecture is not the only way to process queries and there is no perfect query processor.

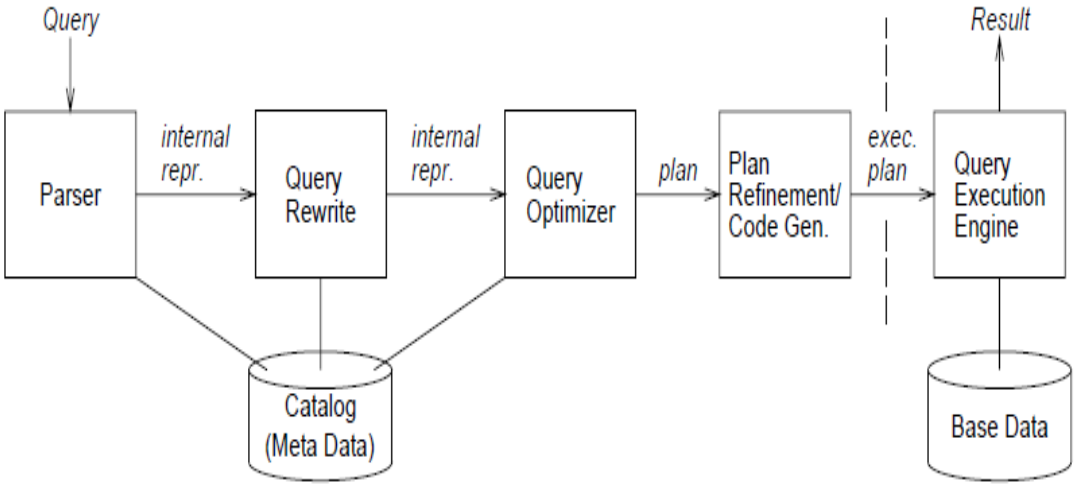

Figure 2.1. Phases of query processing

Each component in Figure 2.1 is briefly explained in [2], first the query arrives to the parser and then it is parsed and translated into an internal representation such as query graph and transferred to the Query Rewriter. The Query Rewriter component

transforms the query in order to carry out optimizations which are regardless of the physical states of the system. The query optimizer carries out optimizations with considering the physical state of the system by making some decisions and these decisions are chosen from the alternative plans that the query optimizer enumerates. Plans specify how to execute the query. Plan Refinement/ Code Generation component transforms the plan generated by the optimizer into an executable plan. Query Execution Engine provides the generic implementations for each operator. Also in Figure 2.1 there is a component called Catalog, it describes the specific characteristics of elements in tables such as the length of the field, the number of columns in a table, the partition schemas and the location of the table.

In distributed query processing many query processing strategies can be generated with varying local processing and communication costs. Query optimization is for finding an efficient way of processing the query with the minimum cost among all query processing strategies.

2.3. Cost Measures and Join Operation

Distributed query processing is the process of retrieving data from different sites. Select, project, join and semijoin operations are the most common relational operations that are performed on database [21, 22, 23]. The join of relation Rwith relation Son attribute A is denoted by R.A=S.A, where Rand Sare joining relations and the attribute A, which is an element of both relation R and S, is the joining attribute of relation Rand S. The join operation is used to combine related tuples from the two relations into single tuples that are stored in the result relation. The only difference between cartesian product and join operation is that the join operation has join condition and according to that join condition related tuples are selected from the two relations. In relational database the join operation is one of the most common operator and very time consuming [5].While implementing join operation each tuple of one relation must be compared with each tuple of the other relation. The cost of a join operation in local database consists of CPU cost and disk I/O cost. CPU cost is the cost of making operation on data in main memory and disk I/O cost is the cost of making disk input output operations. The join operation acceleration methods on local databases focus on to overcome the disk I/O cost and CPU cost.

The cost of an operation in a distributed system is composed of local processing cost and communication cost [1,2]. Local processing cost is composed of CPU cost and I/O cost; communication cost is the cost of transferring data from one site to another through a communication network. In a distributed system, in which databases are geographically separated, the communication cost is the dominant factor through other costs. Proposed works for query optimization in distributed system can be categorized in two main approaches [3]; The first one minimize the cost of data transferred across the network by reducing the amount of transferred data and the second one minimizes the response time of the query by using parallel processing techniques.

Cost estimation for optimizing the query can be done in two different ways; by considering the response time of the query or by considering the total resource consumption needed to process the query. The response time of the query is the time needed to process the query from initiation of the query till taking the answer. Minimizing the response time of the query can be made by using parallel processing techniques. In a local system, the total resource consumption is composed of CPU cost and disk I/O cost.CPU cost is the cost of making operation on data in main memory and disk I/O cost is the cost of making disk input output operations. Minimizing the total CPU cost and total disk I/O cost minimizes the total resource consumption.

Transmission cost is explained as a linear function in [9] as C_0+C_1*X where C_0 is the startup cost for initiating transmission and C_1 is the fixed cost per byte transmitted in time unit and X is the amount of data(usually measured in number of units) to be communicated[1].

2.4. Query Optimization Strategies

Many algorithms were proposed to find the optimal solution for processing a query in distributed system. In this section SDD1 and AHY algorithms are explained, because both of them are query optimization algorithms for point to point wide area network connections and also both of them use semijoin operation as reducer. Semijoin operation is beneficial if its cost does not exceed its benefit. The cost of performing $R \ltimes S$, is the data transmission cost of moving the join attribute from the input relation R to the site where S is located and the benefit is the reduction in transmission cost by reducing the size of relation S [11]. A semijoin is beneficial if cost-benefit<0.

2.4.1. SDD1 Algorithm

SDD1 algorithm is designed to optimize queries in SDD1 distributed database. The algorithm is designed to minimize the data transferred between sites while processing a query. The SDD1 queries are processed as follows [1,10,12];

1. **Query Mapping:** Map a Data language query Q into relational calculus form (called and envelope) that specifies the superset of the database that is required to answer Q.
2. **Envelope Evaluation**: Construct a reducer P which contains a sequence of relational operations. Select a site S such that the cost of computing P and moving the result to S is minimum over all reducer sites.
3. **Query Execution**: Execute Q at S using the data assembled at step 2.

The SDD1 algorithm is derived from an earlier method called "hill-climbing" algorithm, the hill-climbing algorithm does not use any semijoin operation and also does not consider any replication or fragmentation, it was designed for reducing both response time and total processing cost.

The input of the SDD1 algorithm includes the query graph, location of relations and relation statistics [6] and it gives an execution strategy as output. A distributed query processing by SDD1 is as follows [10];

1. Initialize the program P to contain local operations.
2. Repeat

   Add to P profitable non-local semijoins.

   Until no more profitable semijoins are found.
3. Select assembly site and append to P the necessary commands to move reduced data to assembly site.

2.4.2. Algorithm General

One of the most popular and important algorithms suggested for query optimization with minimizing the cost is Algorithm General[3] which was introduced in[4]. Algorithm General has three phases [1,3];

1. **Local Processing Phase:** Involves all local processing operations such as projection and selection.
2. **Reduction Phase:** Involves semijoin reduction and data shipment from one site to another to be reduced.
3. **Final Processing Phase:** All resulting relations are sent to the site where the final query processing is performed.

It is possible to distribute data among different locations in distributed database system. Distributed database system consists of physically separated databases, and these databases act as one local database by using distributed database management system. Query optimization is to find a way to process a query with minimum cost. Processing a query in distributed system consists of local processing cost and transmission cost. Distributed query optimization algorithms focus on reducing the transmission cost rather than reducing the local processing cost. SDD1 and Algorithm General are the two distributed query optimization algorithms that are designed to minimize the transmission cost.

In a distributed system while performing join operation, it is often needed to transfer data between sites, if the final querying site does not contain the relations that participate in final join result. There are many previous works[15][16][13] [17][18] done for reducing the time of processing join operation in distributed system. In the next chapter, some of these works will be stated in detail.

CHAPTER 3

JOIN OPERATION IN DISTRIBUTED SYSTEM

Using Bit Array in optimization of join queries in distributed systems, in which sites are often connected with low bandwidth and strong latency and contain an increasing data volume and consist of high numbers of data sources, has been the topic of many researches [13]. For example if there are two relations, relation R and relation S where both of them are placed at two distinct sites site1 and site2 respectively and a join operation result is needed at site2. Straightforward plan for join operation between relation R and S is to send relation R from site1 to site2 and perform a local join at site2 [14]. Depending on the selectivity of relation R, a high volume of redundant data might be sent from site1 to site2. Redundant data consists of the records which does not take place in the final join result. Cost of data transmission shadows local processing cost and many optimization methods are focused on reducing the size of relations before transferring them to other sites. Semijoin is one of the popular methods for reducing the communication cost and there are many previous works which are based on semijoin operation [15][16][13] [17][18].

Data compression is an effective means for saving storage space and network bandwidth [12]. Rather than sending the actual records between sites, they can be compressed before they are sent. Bit array consists of just 1 and 0 values. The actual data can be encoded into a bit array and then transferred to another site and the receiving site can encode the bit array. While one record in a bit array is just 1 bit the actual records can be dozens of bytes. Therefore transferring the bit array instead of actual records saves communication cost.

However encoding actual records to a bit array and decoding bit array values to actual records consumes some extra local processing cost. It should be carefully considered whether to use bit array in join operation or not. Bit arrays can be beneficial in low bandwidth networks. On the other hand, in fast networks local processing cost may shadow communication cost if bit array is used.

3.1. Join Operations without Compressing Data

Semijoin operation can be used to reduce the amount of data transferred over the network in order to reduce the communication cost [15]. Semijoin from relation R to relation S (S⋉R) is processed as follows [13];

1. Project join attributes of R
2. Send projected values to site2
3. Make join operation between projected values of R and table S

After performing join operation at step 3, if all tuples of relation S is not going to be participate in final join result between table R and S, the volume of resulting relation S' is going to be less than table S because S' is just going to contain the tuples that are going to take place in the final join result between table R and S. So, if the final query site is at site1, sending S' to site1 from site2 is going to be cheaper than sending the relation S from site2 to site1.

An example is proposed in Figure 3.1. In this example, relation R is placed at site 1 and relation S is placed at site 2 and join operation between relation R and S is needed at site1. Semijoin operation can be used to reduce amount of data transfer before performing join operation. First, the projection of join attributes of relation R is taken and then shipped to site2. After, site2 receives projected values it makes join operation between projected values and relation S to eliminate redundant records at relation S. Then, the reduced relation S is transferred to site. After site1 receives the reduced relation S, it performs join operation between relation R and reduced relation S.

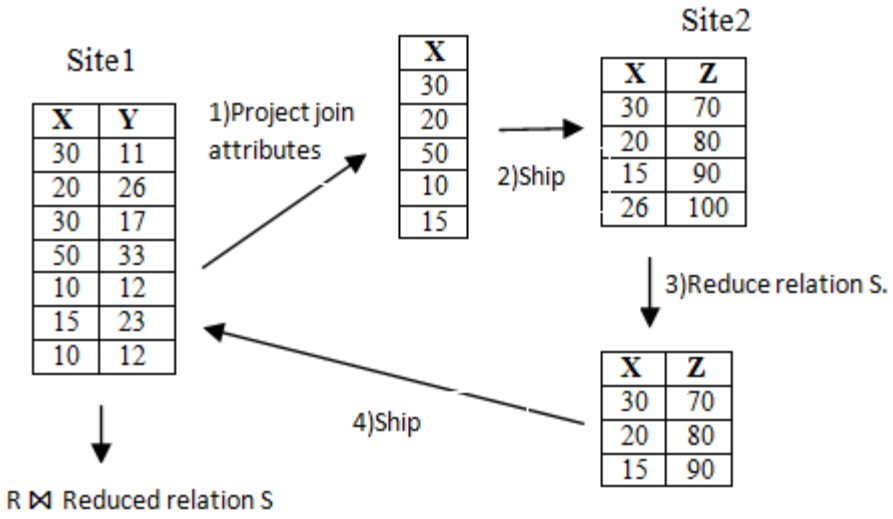

Figure 3.1. Using semijoin operation for R⋈S

2-Way-Semijoin is an extension of semijoin operation. Semijoin operation has one direction, which means it eliminates redundant tuples from one relation. It is possible to eliminate tuples from both relations by using 2-way-semijoin [3, 19] operation. 2-way-semijoin [19] is designed to eliminate tuples from forward and backward direction. . The additional reduction is equivalent to the reduction of two symmetric and sequential semijoins [3].The algorithm of 2-way semijoin can be explained as below [19];

1. Project join attributes of R.
2. Send projected values to site2.
3. Make join operation between projected values of R and table S to reduce table S. S'=S⋈$\prod_X$(R).
4. Partition projection of join attributes of table R ($\prod_X$(R)) into Rm[x] and Rnm[x] where Rm[x] = $\prod_X$(R)⋉ S and Rnm[x] =$\prod_X$(R) −Rm[x].
5. Send either Rm[x] or Rnm[x] according to which has small size.
6. Reduce table R by using Rm[x] or Rnm[x]. If Rm[x] is used then tuples whose attributes X are not matching any of Rm[x] are eliminated. If Rnm[x] is used then tuples whose attributes X are matching any of Rnm[x] are eliminated.

The first three steps of the algorithm are same with thesemijoin operation and it is forwards reduction phase. In the forth step a semijoin operation between projected values of relation R and relation S is performed to eliminate redundant records from the

projection of relation R and the result is saved in an array called Rm[x]. Then a second array is created which contains the redundant tuples of the projection. At the fifth step from Rm[x] and Rnm[x], the one whose size is smaller is sent from site1 to site2. At the sixth step after site1 receives Rn[x] or Rnm[x], it reduces relation R. Most of the query processing algorithms based on semijoins can be modified to use 2-way-semijoin [5].

An example of join operation by using 2-way-semijoin is shown in Figure 3.2. In the example it can be seen that the join operation result is needed at site2 and 2-way-semijoin operation eliminates redundant tuples from both relation R and relation S.

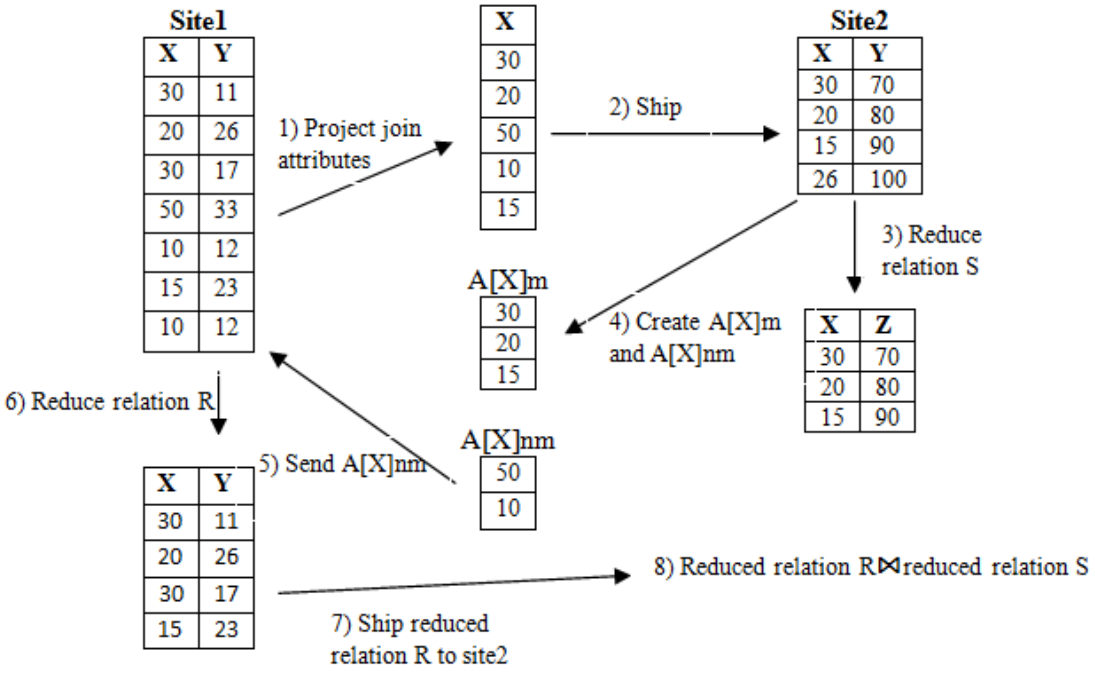

Figure 3.2. Using 2-way-semijoin operation for relation R and S

3.2. Join Operations with Compressing Data

In semijoin and two-way-semijoin operation a small part of a relation which is projection of join attributes is send from one site to another. Rather than sending the actual records from one site to another they can be compressed before sending between sites. Compressing the data before sending it between sites can reduce the

communication cost. **Bloom Filter** is a data structure was first proposed by Burton H.Bloom[16]. It consists of an array of n bits and k number of independent hash functions F={f1,f2…fk} and the result of each hash function is in the range of 1 through n. [20] Initially the bits in the array are set to 0 and then to fill the array each element is put in to each hash function and the result obtained from the functions shows which address of the bit array is set to 1.

As an example suppose that there is a set S={a,b,c} and there is 4 hash functions f1,f2,f3,f4 and the array in bloom filter structure is 16 bits. First it is needed to set each bit to 0 to indicate that bloom filter is empty, then each hash function is going to be used with each element in the set and the value of the bits are going to be set to 0 or 1according to the result of the hash functions. And suppose the values of each function are;f1(a)=3,f2(a)=2,f3(a)=14,f4(a)=5,f1(b)=13,f2(b)=12,f3(b)=4,f4(b)=11,f1(c)=9,f2(c)=7, f3(c)=15, f4(c)=6, then the resulting array will be as in Figure 3.3.

0	1	1	1	1	1	1	0	1	0	1	1	1	1	1	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Figure 3.3. Bloom array for set S

The **Bloom-join** algorithm which outperforms the basic semijoin algorithm, encodes the projected values by using bloom filters and then sends the bloom filter to the other site, and the site receiving the bloom filters uses the bloom filter to reduce the relation it contains and it sends the reduced relation to the site where the join operation is going to occur. The algorithm of Bloom-join is as below;

1. Produce a Bloom filter BFR for projection of join attributes of relation R.
2. Send BFR from site1 to site2.
3. Filter the records in table S that are going to participate in join operation by using BFR
4. Send reduced relation S from site2 to site1.

In Figure 3.4 an example for Bloom-join operation is given. In the example there is 2 hash functions(f1,f2) used and the results of each function is ;f1(30)=3, f2(30)=7, f1(20)=5, f2(20)=1, f1(50)=4, f2(50)=10, f1(10)=8, f2(10)=2, f1(15)=6, f2(15)=11. At step 2, the projected join attributed are put into a bit array by using the hash functions

and then transferred to site2. When site2 receives the projected values it decodes the bit array by using the same hash functions and reduces relation S.

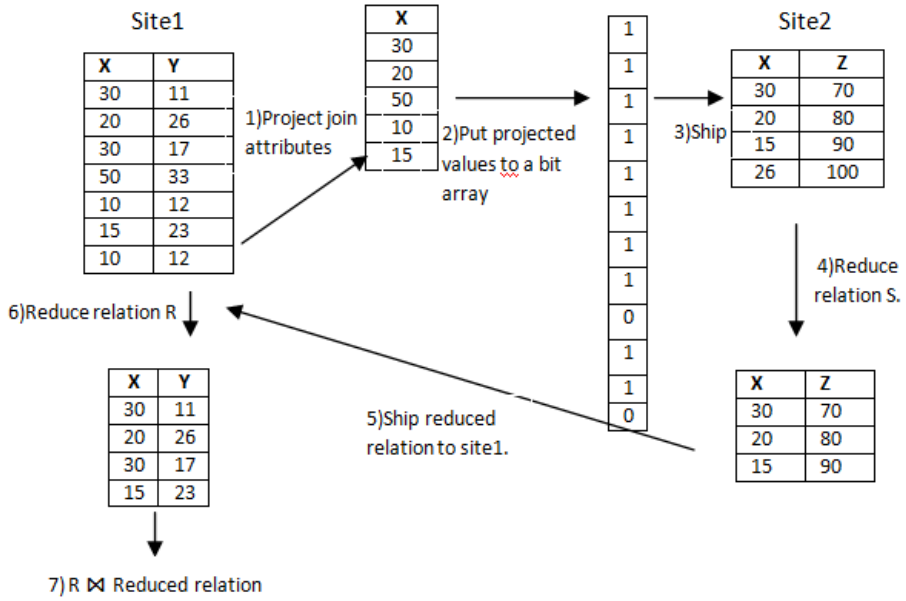

Figure 3.4. Bloom-join operation between relation R and S

The difference between semijoin and bloom-join is that the bloom-join encodes the actual records by using hash functions to values between 1 and 0 before sending them and also the receiving site decodes the received data by using the same hash functions. Instead of sending the actual records between sites bloom-join transfers just values 1s or 0s which results in reducing the amount of the data transferred between sites. However due to nature of hash functions, it is known that hash functions may result in hash collisions, and this can result in data loss.

[13] Introduces using **algebraic signatures** for semijoin based operations. An algebraic signature which can be used to identify a tuple and different signatures is a few bytes of strings. Algebraic signatures can prove the inequality of the contents. The same signatures indicate the equality with overwhelmingly high probability and they are very efficient for string matching [13]. The algorithm for using algebraic signatures for semijoin operation is given below;

1. Generate algebraic signatures for join attributes of relation R, Sign j(r).
2. Insert Sign j(r) into a hash table Hsign.
3. Send Hsign to site2.
4. Generate algebraic signatures for join attributes of relation S, Sign j(s).

5. Compare the algebraic signatures of both table and reduce relation S by eliminating non matching algebraic signatures.
6. Send reduced relation S to site1.

At step 1 algebraic signature of each join attribute is generated and then at step two they are put into a hash function in order to put in a bit array. Then at step 3 the bit array is send to site2 and then at the forth step, algebraic signatures of join attributes of relations S is generated. At step five if the value in bit array is not 0, then it is decoded and algebraic signatures of join attributes of relation R and S is compared. Comparing algebraic signatures rather than the actual data is faster because signatures are 4 bytes long where string they signed could be dozen bytes [13]. The only difference between bloom-join and algebraic signature based semijoin operation is, algebraic signatures of join attribute values are put in a bit array, it gives an advantage in comparing the join attribute values of two relations.

Virtual join operation [17] suggest in distributed query processing local processing cost should not be neglected. It points out that knowledge about the final join result is not balanced at each site. As an example, after site 1 projects join attributes of relation R and send them to site2, site 2 will have knowledge about which tuples are useful at its own site while site1 still knows nothing about the final result. Virtual join operation generates a representation of final result which is called virtual result. The size of the virtual result is much smaller than the real result, and it gives information about the cardinality of the real result. The virtual result is generated by exchanging some data between sites. In Figure 3.5 the virtual result is shown. Virtual result has three fields: the first field is the join attribute field and the other two fields gives information about the number of the useful tuples at relation R and S.

Site1
Relation R

X	Y
30	11
20	26
30	17
50	33
10	12
15	23
10	12

Site2
Relation S

X	Z
30	70
20	80
15	90
26	100

Virtual Result

X	R	S
30	2	1
20	1	1
15	1	1

Figure 3.5. Virtual result of relation R and relation S

Virtual table can build by bloom-join [18] as explained below;

1. Hash join attributes of relation R and S.
2. R and S sent the bloom-filters to each other.
3. R and S exchange the number of distinct join attributes.
4. The relation that has the smaller number sends the join attributes without redundant ones and with the number of tuples associated with each join attribute value, to the other site.
5. Suppose at step four R send data to S, then S will make a hashed local join to fully reduce its tuples.
6. S creates a vector in which each element indicates the number of tuples in S with corresponding join attribute value.
7. Site 1 builds Virtual Table

By using the Virtual Table, the site,whose going to make the final join operation can be chosen. The Virtual Join operation proposes that final join operation is not have to make at the assembling site. It is clear that Virtual Join reduces the communication cost and local processing cost by handling the join operation at the site which has more redundant tuples than other sites. However, an additional transfer cost should be considered if the result of the join operation need to positioned at the site which has low redundant tuples than other, because the result of the join operation should be send from one to another.

Positionally Encoded Records Join (Perfjoin) [3, 17] is an extension of 2-way-semijoin operation, it reduces both relation as in 2-way-semijoin. In 2-way-semijoin after site2 receives projection of join attributes of relation R it reduces relation S and this is called forward phase, then sends Rm[x] or Rnm[x] to site1 and this is backward phase. Perf-join operation reduces the communication cost of backward phase of 2-way-semijoin operation, by sending a bit vector rather than sending the actual records. Perf-join operation does not use any hash function as in bloom-join operation to create the bit vector, it makes tuple scanning and this prevents the hash collisions.

The PERF of relation R with respect to S is denoted by PERF(R) and it is a vector at the size of the cardinality of relation R [17]. In Figure 3.6 PERF(R) and PERF(S) is shown by an example. The value 1 in the bit vectors indicate that the tuple in the same position is going to be participate in the final join operation result. In PERF(S), the first three values are 1, this means first three tuple at relation S is going to participate in the final join result and in PERF(R) first, second, third and forth values

are 1 and this means the first, the second, the third and the forth tuples in relation R is going to participate in the final join operation result.

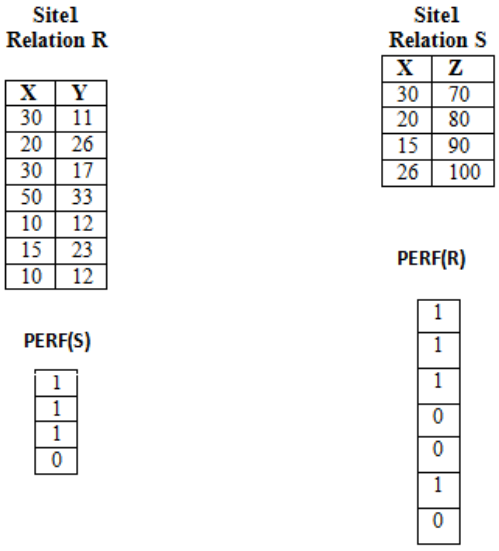

Figure 3.6. PERF(R) and PERF(S)

The algorithm of PERF join is given below [3];

1. Project R on a joining attribute and get P_R.
2. Ship P_R to site2.
3. Reduce S by a semijoinwith P_R.
4. Send back to site1, a bit vector (the PERF) that contains one bit for every tuple in P_R and in the same order.

The first three step of the algorithm is the same with 2-way semijoin operation. The difference is PERFjoin operation sends back to site1 a bit vector rather than sending actual records and this reduces the amount of data transfer between sites.

PERF join is illustrated with an example in Figure 3.6. In the example the final join operation is made at Site2. At the first step projection of join attribute values of relation R is taken without eliminating the duplicated values because the position of each record is important to generate PERF(R).

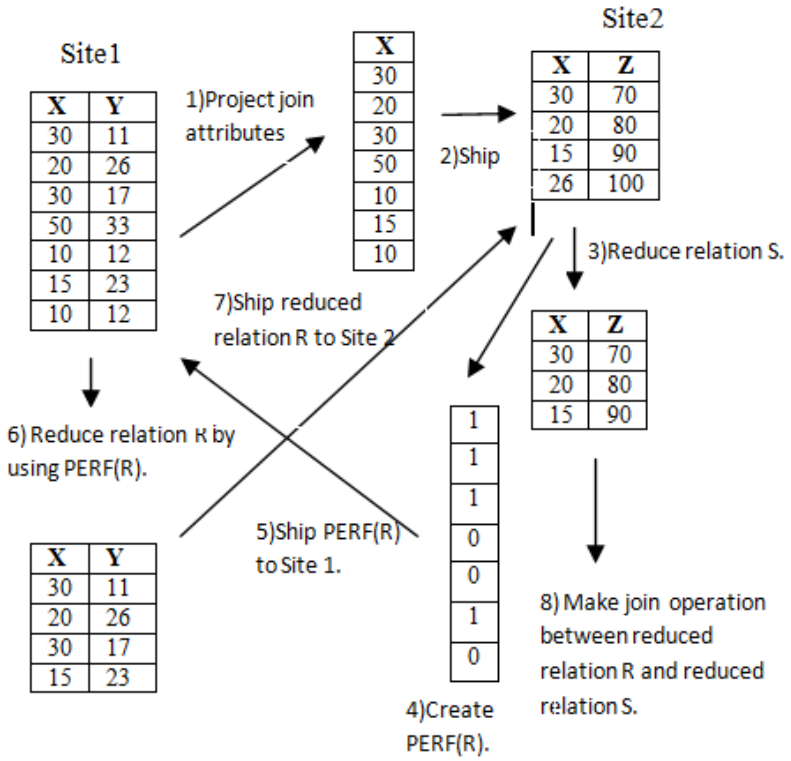

Figure 3.7. PERF join between relation R and relation S

PERF join operation overcomes the hash collision problem by not using hash function and this prevents the data loss. However, a relation may consists of many duplicated join attribute values, and if those duplicated records are not eliminated significant amount of redundant data can be transferred between sites. [17] suggests that duplicate elimination can be done by sorting operation, but it is known that sorting operation is a cost effective operation and this is going to increase the local processing cost.

This chapter is aimed at discussing related works in the area of join operation in distributed database system. In this chapter join operations in distributed database system are grouped in two categories: join operations without compressing data and join operations with compressing data. A summary of the related work is presented on Table 3.1. In order to compress the data, it is needed to decode the records before sending them and also at the receiving site it is needed to decode the encoded records. Encoding and decoding records adds extra local processing cost at both sites and it is a time consuming work. Join operations with compressing data becomes advantageous if the connection speed between sites is low. The extra local processing cost, which is

appeared while making encoding and decoding operations on data, is going to be negligible because the transmission cost becomes the most cost effective work.

Bloomjoin operation puts join attributes values into a bit array by using hash functions and sends the bit array to the other site rather than sending the actual records however it is not possible to ensure that there will not be any data loss while decoding the bit vector at the receiving site. On the other hand PERFjoin operation puts the records into a bit vector as in bloomjoin operation, but it does not use any hash functions to create the bit vector and this ensures that there will not be any data loss. Although PERFjoin operation prevents data loss while encoding and decoding the records it does not consider duplicated join attribute values and does not make any duplicate elimination.

In the next chapter a new semijoin based join operation named Distinct Encoded Records (DERjoin) is introduced. DERjoin operation is designed for both considering the low bandwidth connection between sites and duplicated join attribute values.

Table 3.1. Summary of join operations

	Advantage	**Disadvantage**	**Possible to Execute in Parallel**	**Compresses Data**	**Eliminates Duplicated Join Attributes**
Semijoin	Low local processing cost.	Reduces just one relation, expensive communication cost.	N	N	Y
2-Way-Semijoin	Reduces both relations.	Expensive communication cost	Y	N	Y
Bloom-Join	Reduces communication cost.	Extra local processing cost, collusion.	Y	Y	Y
Algebraic signature based Semijoin	Fast in matching join attributes, reduces communication cost.	Extra local processing cost, collision.	Y	Y	Y
Virtual Join	Considers both local processing and communication cost.	Extra local processing cost, collision.	Y	Y	Y
Positionally Record Filters join	Does not use any hash function, reduces communication cost.	Extra local processing cost.	Y	Y	N

CHAPTER 4

A NEW SEMIJOIN BASED JOIN OPERATION

In this chapter a new Semijoin based join algorithm which is called Distinct Encoded Records Join operation is introduced. Then the DERjoin algorithm is briefly described and an example is given to show how the DERjoin operation performs. It is assumed that there is no data replication or fragmentation so each site stores one relation and also it is assumed that the network which connects the sites is a point to point wide area network and sites are separated geographically. It is known that communication cost shadows local processing cost and it is the dominant factor that effect the join processing time in geographically separated databases. Because of that rather than the local processing cost, the amount of the data transferred between sites is reduced in DERjoin operation.

4.1. Distinct Encoded Records Join

Distinct Encoded Record join (DERjoin) is designed to eliminate duplicated values in the projection of join attribute values to reduce the communication cost and to prevent the loss of data while creating the bit vector. The algorithm is designed for geographically separated sites and it is assumed that sites are connected with low bandwidth network connection. For example if there are two relations, relation R and relation S which are located at two distinct sites, site1 and site2 respectively and both relations have a common join attribute and the final join operation is going to be performed at site2, with another meaning the assembling site is going to be site2. A small portion of relation R is going to be shipped to site2 to reduce both relations R and S, and this small portion of data is the projection of join attributes of relation R.

Bit vector consist of 1s and 0s and in a distributed join operation they can give information about which tuples of a relation are going to participate in the final join result. To create the bit array there is no hash functions used to prevent hash collisions, tuple scanning is made to create the bit vector.

The algorithm of DERjoincan be explained as below;

1. Project distinct join attributes of R.
2. Ship distinct join attributes from site1 to site2.
3. Reduce S by a semijoin operation with distinct projection of join attributes of R.
4. Send back to site1, a bit vector that contains one bit for every record in distinct projection of join attributes in the same order.

At the first step of the algorithm distinct join attributes of relation R is taken to eliminate duplicated records in the projection of join attributes of relation R, and then those projected values are shipped to site2. After site2 receives the projected values it makes a semijoin operation between projected values and relation S to reduce the size of relation S. After relation S is reduced with a semijoin operation, a bit vector is created at site2 whose length is equal to the number of distinct projected join attributes of relation R. If the jth bit of the bit vector is 1 then it means the jth record of projection of table R is going to participate in the final join result and if the jth bit of the bit vector is 0, then it means jth record of projection of table R is not going to participate in the final join result. The bit vector just gives information about the distinct projection of join attributes of relation R. It is not possible directly to understand which tuples of relation R are going to participate in the join result. The key point is, the bit vector is always going to give idea about the first tuple of relation R, if record at bit vector at the first address is set to 1, it means first tuple at table R is going to be participate in join result else it is not.

Another bit vector *bitvector2* at site1, whose length is equal to number of rows of relation R, should be created to keep which tuples are traversed. So, when site1 receives the bit vector (*bitvector1*) which is sent from site2, it is going to create a second bit vector (*bitvector2*) whose length is equal to the number of rows at relation R. Initially each record in *bitvector2* is going to be set to 0. The value 0, in the *bitvector2* indicates that the tuple in relation R with the same address in *bitvector2* is not traversed and the value 1 in *bitvector2* indicates that the tuple in relation R with the same address in *bitvector2* is traversed. After *bitvector2* is created, the first element of the *bitvector1*is going to be checked and if it is 1, then tuples, whose join attribute value is equal to the join attribute value of the first tuple of relation R are going to be selected and also the bits in *bitvector2* with the same address of the selected tuples from relation R are going

to set to 1. If the first element of the *bitvector1* is 0 than there will not be too many differences, the only difference will be the tuples of relation R whose join attribute values equal to the join attribute value of the first tuple would not be selected, and the bits in *bitvector2* with the same address of the tuples in relation R that contains the same join attribute value with the first tuple of relation R, are going to be set to 1. After all, there is a need to find the first element at *bitvector2* whose value is 0. The first 0 valued bit at *bitvector2* gives information about which tuple at relation R is not traversed.

Figure 4.1 illustrates the *Distinct Encoded Records join* operation between relation R and S where the assembling site is site2 and the common join attribute is "attribute1". At first, the distinct projection of join attributes are taken at site 1 and then shipped to site 2. After site2 receives the projected values, it makes a semijoin operation between relation S and projected values to reduce the size of relation S. Then, it creates a bit vector whose length is equal to the number rows of projected values and then bit vector is shipped to site1. Site1 reduces relation R by using the bit vector and ships the reduced relation R to site2. Site2 makes the join operation between reduced table R and reduced table S.

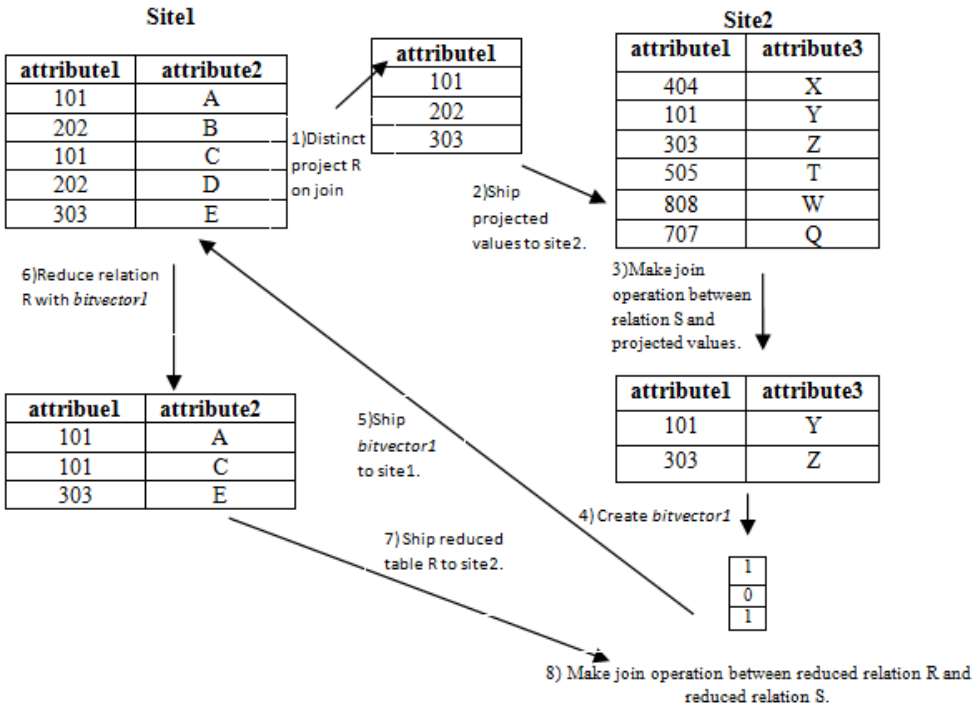

Figure 4.1. DERjoin between relation R and relation S

The Derjoin algorithm eliminates tuples at both relations before making the final join operation. The elimination of redundant tuples from relation S is called forward reduction phase and the elimination of redundant tuples from relation R is called backward reduction phase. The next sub section briefly explains backward reduction phase.

4.2. The Forward Reduction Phase

It is assumed that both the forward and backward reductions are done at the sites storing the two joining relations. The forward reduction phase is for reducing the relation S, and in the forward reduction phase a semijoin operation is performed at site2. The semijoin operation between relation S and distinct projected values of R eliminates the redundant tuples in relation S. In Figure 4.2 the result of the semijoin operation is illustrated.

Projection of join attributes of relation R.

attribute1
101
202
303

Relation S.

attribute1	attribute3
404	X
101	Y
303	Z
505	T
808	W
707	Q

Result of S ⋉ projection of join attributes of relation R.

attribute1	attribute3
101	Y
303	Z

Figure 4.2. Reducing relation S

The result of the semijoin operation that is performed between relation S and the projected values is the set of all tuples in S, for which there is a tuple in projected values that is equal on their common attribute names. Reducing the size of relation S is going to reduce the time needed to perform the final join operation between relation R and relation S. After the reduction process, a bit vector which gives information for each record in the projection of relation R is created. The jth record at the bit vector is set to 1 if the jth record at the projected values is going to participate in the final join operation result and it is illustrated in Figure 4.3.

Projection of join attributes of relation R.

attribute1
101
202
303

The bit vector created at site2.

1
0
1

Figure 4.3. The reduction information of projected join attributes

In Figure 4.3 the first and the third records in the bit vector is 1 and this means the join attribute values 101 and 303 are going to participate in the final join result. The second record in the bit vector is 0 and this means the join attribute value 202 is not going to participate in the final join operation result. After the bit vector is created, it is going to send from site2 to site1 and site1 is going to reduce relation R by using the bit vector and this reduction operation is called backward reduction phase.

4.3. The Backward Reduction Phase

In the backward reduction phase relation R is reduced at site1.The bit vector (*bitvector1*) that is created at site 2 gives information about the projection of join attributes of relation R. It is not possible to directly understand which tuples of relation R participate in the final join operation result by looking at*bitvector1* because the position of each element in *bitvector1* is same with the projected values not with the relation R. If duplicated join attributes exists in relation R than the size of *bitvector1* will be different than the cardinality of relation R.

After site 1 receives *bitvector1* another bit vector (*bitvector2*) whose length is equal to cardinality of relation R is needed to be created to find out which tuples of relation R are traversed. Each record of the *bitvector2* gives information about each tuple of relation R.If the jth bit of the *bitvector2* is 1 then it means jth record of relation R is traversed and if the jth bit of the bit vector is 0 then is means jth record of relation R is not traversed.

According to the example in sub section 4.1, after site 1 receives *bitvector1* it reduces relation R as explained below;

- A bit vector (*bitvector2*), whose length is equal to number of rows of relation R (number of rows of table R is 5), is created and each row is set to 0.

- The first row of bit vector received from site1(also called it *bitvector1*) checked and its value is 1 so this means tuples whose *attribute1* value is 101 participates in R⋈S result, and those tuples are selected and rows 1 and 3 at *bitvector2* are set to 1.
- The second row of *bitvector1* is checked and its value is 0 and the first 0 valued record at *bitvector2* is positioned at the 2nd row, this means join attribute value positioned at 2nd row at relation R is not going to participate in join result, by another meaning tuples whose *attribute1* value is 202 does not participate in R⋈S result, and 2nd and 4th rows of *bitvector2* are set to 1.
- Third row of *bitvector1* is checked and its value is 1 so this means tuples whose *attribute1* value is 303 participate in R⋈S result, and those tuples are selected and fifth row at *bitvector2* is set to 1.

The backward reduction phase finishes when there is no 0 valued bit remains in *bitvector2*. The selected tuples from relation R till all records at *bitvector2* are set to 1, are the tuples of relation R that are going to participate in the final join operation result.

When the number of duplicated join attribute values at relation R increases, the size of the projected values of relation R and the length of the *bitvector1* decreases. Because the algorithm just sends the distinct projection of join attribute values and also length of the bit vector created at site 2 (*bitvector1*) has the same length with distinct projection of join attribute values. The decrease in the size of projected values and *bitvector1* results in a decrease in the volume of data that is send between site1 and site2.

DERjoin operation is designed to reduce the communication cost in geographically separated distributed systems. It adds some extra local processing cost to both sites for eliminating duplicated values in the join attributes before sending them to other sites for creating the bit vectors and also for decompressing the data in the bit vector. It is not advantageous to use DERjoinoperation in a high bandwidth network becausein a distributed system, in which sites are connected with high speed connection, communication cost is not going to be the basic factor that affects the total time needed to make join operation between two relations. In such a system both local processing and communication costs should be considered.

CHAPTER 5

PERFORMANCE EVALUATION

In this performance evaluation study Distinct Encoded Records join(DERjoin) is compared with Positionally Encoded Records join (PERFjoin) operation. PERFjoin and DERjoin operations are similar in that they both have forward and backward phase. In the forward phase, the projection of the join attributes is send from one site another and in the backward phase, a bit vector is created and send from one site to another. A second similarity is that both operations are focus on reducing the communication cost while neglecting the local processing cost. They also have differences, DERjoin eliminates duplicated values from the projection of join attributes and also the bit vector that is created in the backward phase of DERjoin operation might not directly give information about which tuples of the relation are going to be participate in the final join operation result.

5.1. Experimental Setting

The performance is conducted on two computers that are connected with 10Mbps local area network. Software and hardware features of each computer are similar. Each computer has Pentium(R) DualCoreE6300@2.8Ghz 2.8Ghz and 2GB RAM of Windows7 Home Premium. Both DERjoin and PERFjoin are implemented in Visual Studio .NET 2010 Ultimate Edition and data transfer between each computer is handled by using .NET Remoting. The full implementation source code is presented in Appendix A.

Data sources used in the performance test are text files named tableR.txt and tableS.txt, tableR.txt is stored at computer1 and tableS.txt is stored at computer2 and the final join operation is handled at computer2. Both tables have two fields and except the second experiment, the cardinality of table R is 10.000 and table S is fixed to 20.000. The field names of table R is attribute1 and attribute2: Field names of table S is attribute1 and attribute3. Common join attribute name at both tables is attribute1. Each byte of attributes are randomly selected from ‘a’ to ‘z’. Four different experiments are

carried out and the characteristics of the relations are shown in table 5.1. In the first experiment attribute1 is 10bytes of data, attribute2 and attribute3 are 20bytes of data. Selectivity of table R is fixed to 0.5 and the number of distinct join attribute values of table R is varied from 10.000 to 2000. In the second experiment cardinality of relation R is varied from 5.000 to 40.000 and percentage of unique join attributes in relation R is varied from 20% to 50%, and the selectivity of table R is fixed to 0.5.In the third experiment, length of attribute1 is varied from 100bytes to 800bytes of data, while length of attribute2 and attribute3 are fixed to 20bytes of data, and the selectivity of table R is fixed to 0.5 and distinct join attribute values fixed to 5.000. In the forth experiment the size of attribute1 is 10bytes of data, and size of attribute2 and size of attribute3 are 20bytes of data. Distinct join attributes in table R is fixed to 5000 and selectivity of join attributes are varied from 0.1 to 1 in table R.

Table 5.1. Characteristics of datasets

Experiment Number	**Cardinality of relation R**	**Cardinality of relation S**	**Selectivity of relation R**	**Distinct Join attributes in relation R**	**Size of attribute1(bytes)**
Experiment1	10.000	20.000	0.5	2.000 to 10.000	10
Experiment2	5.000 to 40.000	20.000	0.5	20% to 50%	10
Experiment3	10.000	20.000	0.5	5.000	100 to 800
Experiment4	10.000	20.000	0.1 to 1	5.000	10

5.2. Varying Number of Distinct Join Attributes of Relation R

In this performance test execution time of join operation on relation R and relation S is compared while number of distinct join attributes of relation R is not fixed. Nine tests are performed. At the first test the number of distinct join attribute values are 10.000, then at each test the number of distinct join attribute values are decreased 1.000 till the value is reached to 2.000. DERjoin eliminated the duplicated values from the projection of join attributes before sending them, therefore it is supposed that DERjoin operation will be beneficial than PERFjoin operation while the duplicated values increase. The aim of this performance test is to investigate whether there is any significant difference between DERF join and PERFjoin if the ratio of duplicated join attributes in relation R increases.

Figure 5.1, illustrates join operation execution times of DERjoin and PERFjoin when number of distinct join attributes of relation R varies from 10.000 to 2.000. It can be seen from Figure 5.1 that DERjoin is advantageous when the number of duplicated join attributes increase. Execution time of DERjoin nearly reaches PERFjoin when the number of distinct values is 6.000 and after 6.000 distinct join attributes; DERjoin is more advantageous than PERFjoin operation.

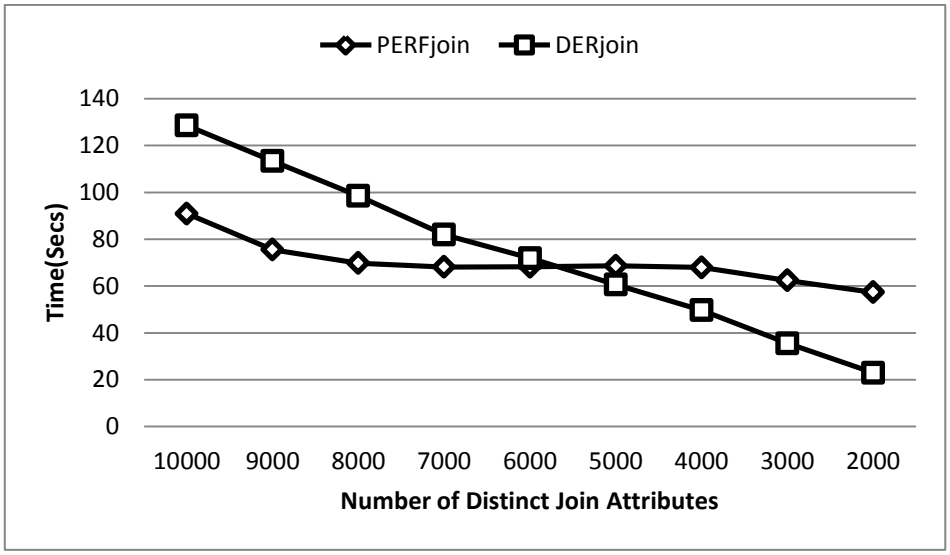

Figure 5.1. Execution time, varying number of unique join attributes in relation R

The total number of bytes transmitted between site1 and site2 for each experiment is displayed in Figure 5.2. DERjoin operation becomes advantageous when the number of duplicated join attributes increase at relation R, because it just sends the distinct values in the forward phase and this reduces the size of data transferred from site 1 to site2. Also, the size of the bit array that is created at site 2 is going to decrease if the cardinality of projection of join attributes decreases, because the length of the bit array created is equal to the cardinality of projected values send from site1 to site2. Results of these tests indicate that when the number of duplicated join attributes of relation R increase then the total number of data transfer between site1 and site2 decrease.

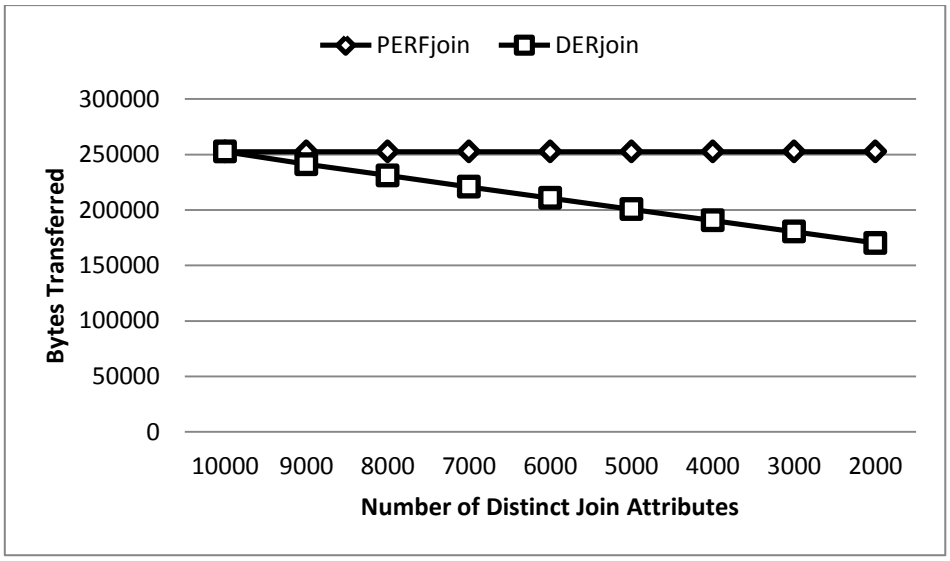

Figure 5.2. Total bytes transmitted varying number of distinct join attribute values of relation R

5.3. Varying Number of Distinct Join Attributes Values and Cardinality of Relation R

In this performance test execution time of join operation on relation R and relation S is compared while number of distinct join attributes and cardinality of relation R is not fixed. Eight tests are performed and at each test cardinality of relation S is fixed to 20.000 and join operation selectivity of relation R over relation S is fixed to 0.5. At the first test cardinality of relation R is 5.000, then at each test cardinality of relation R is increased 5.000 till it is reached to 40.000. Also at each test ratio between distinct join attribute values over cardinality of relation R is set to 0.5, 0.35 and 0.2 with another meaning percentage of total number of unique join attribute values are set to 50%, 35% and 20%. The aim of this performance test is to investigate whether there is any significant difference between DERFjoin and PERFjoin operations while the cardinality of relation R increases and the ratio of distinct join attribute values of relation R changes.

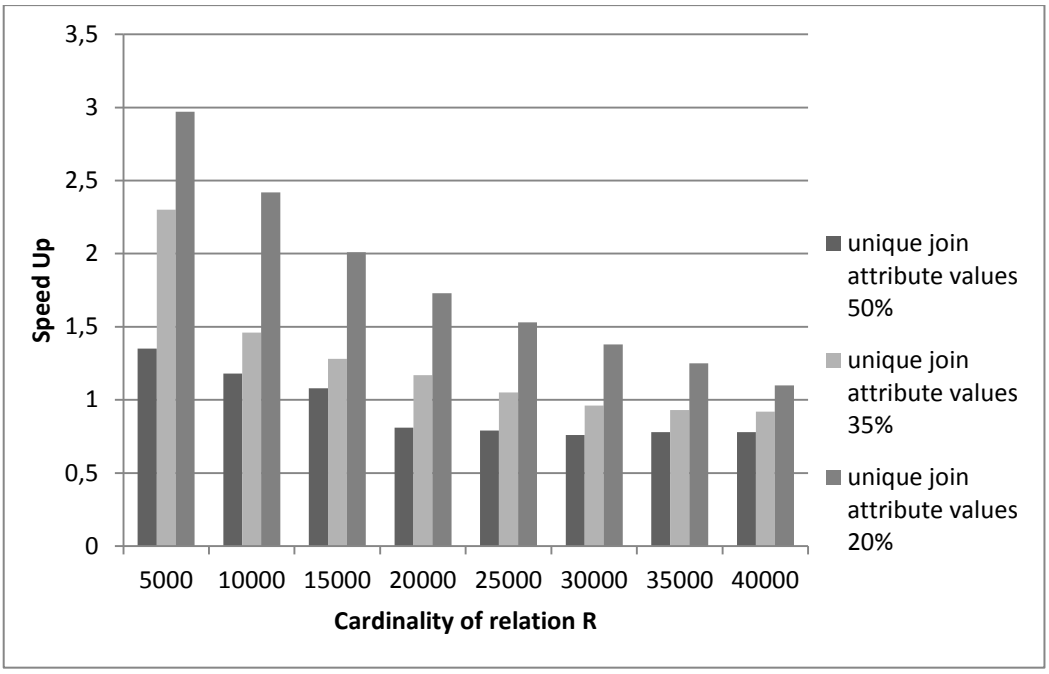

Figure 5.3. Speed up in execution time

Figure 1, shows how DERjoin operation speeds up the query execution time. The speed up is calculated as below;

$$speed\ up = \frac{\text{Execution time of PERFjoin operation(sec.)}}{\text{Execution time of DERjoin Operation(sec.)}} \quad (5.1)$$

Speed up is the ratio between total execution time of PERFjoin operation over DERjoin operation when the cardinality of join attributes of relation R varies from 5.000 to 40.000 and the percentage of unique join attribute values of relation R vary between 50%, 35% and 20% . If speed up is greater than 1 then it means execution time of DERjoin is less than execution time of PERFjoin operation. It can be seen from Figure 1 that DERjoin is advantageous at each test when the ratio of unique join attribute values are 0.2. When the ratio of unique join attributes are 0.35, DERjoin operation is advantageous while the cardinality of relation R is 5.000, 10.000, 15.000, 20.000 and 25.000. Also DERjoin operation is advantageous when the ratio of unique join attribute values is 0.5 and the cardinality of relation R is 5.000, 10.000 and 15.000.

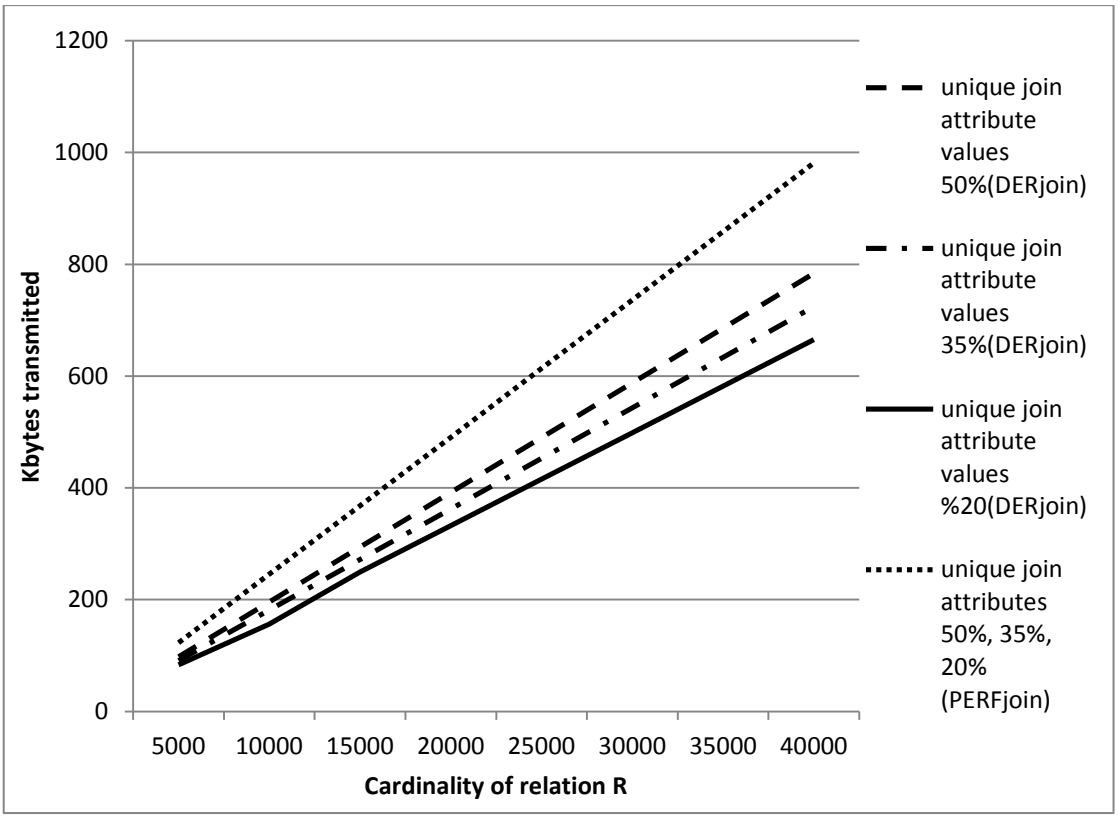

Figure 5.4. Total Kbytes transmitted between computer1 and computer2

Figure 5.4, shows the total Kbytes transmitted between computer1 and computer2 at each test. PERFjoin operation sends same amount of data while the cardinality is fixed and unique join attribute values of relation R varies from 50% to 20% , because it does not make any duplicate elimination. In DERjoin operation at each test total Kbytes transmitted between computer1 and computer2 decreases while the ratio of unique join attribute values of relation R decreases, because it eliminates the duplicated records from projection of relation R before transmitting them to computer2.

At each test ratio between total execution time of PERFjoin operation over DERjoin operation decreased because local processing cost of DERjoin operation increases more than PERFjoin operation while the cardinality of relation R increases.

5.4. Varying Size of a Join Attribute Value of Relation R

In this performance test again the cardinality of relation R is fixed to 10.000 and the cardinality of relation S is fixed to 20.000 as explained before. The difference between experiment 1 and experiment 2 is that while the length of join attribute values

are varied from 100 bytes to 800 bytes of data in experiment 2, in experiment 1 the length of join attribute values are fixed to 10 bytes of data. This test is for showing what will change if the data volume of join attributes changes and the results of this experiment are shown in Figure 5.5. Figure 5.5 shows how many seconds it takes to execute join operation between relation R and relation S by using DERjoin and PERF join operation.

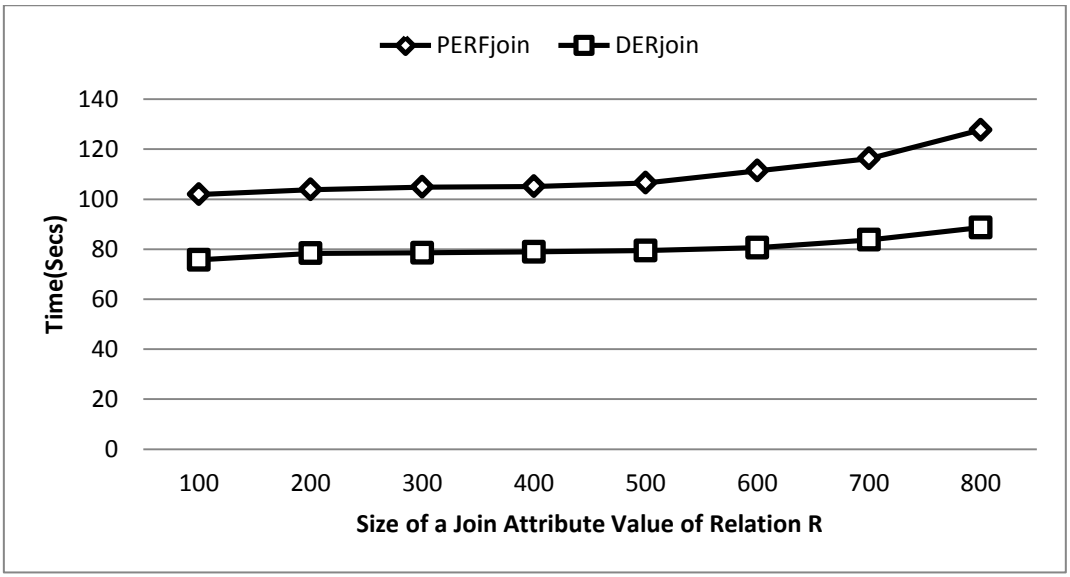

Figure 5.5. Execution time varying size of join attributes value of relation R

The size of data in bytes transferred from site1 to site2 at each test is shown in Figure 5.6. The rate of the data sent from site1 to site2 increases at each step of this test because the size of join attribute values are increase at each test and this adds some extra data to the total number of the data transmitted.

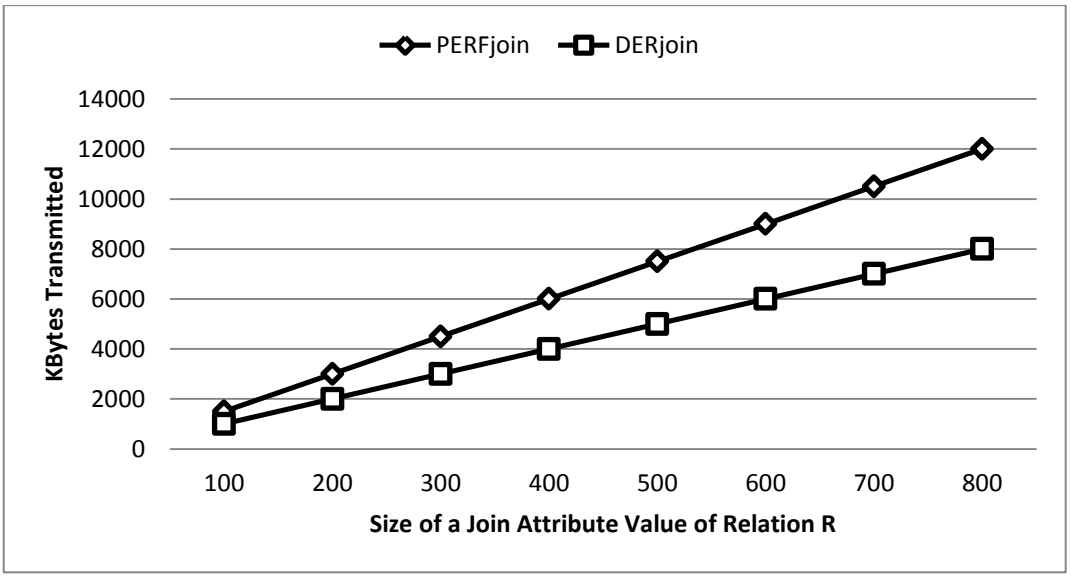

Figure 5.6. Total Kbytes transmitted varying size of join attributes of relation R

5.5. Varying Selectivity of Relation R

In this sub section, distinct join attribute values are fixed to 5.000 and selectivity of relation R is varied from 1 to 0.1. This test is for studying the execution time of DER-join and PERFjoin when the selectivity is not constant. The result of the test is shown in Figure 5.7. The maximum value for the selectivity is 1, which means all tuples of relation R are going to participate in the final join operation result, and the minimum value for the selectivity is 0.1, which means %10 of the tuples of relation R are going to participate in the final join result. When the selectivity of relation R increases, the local processing cost of the final join result increases at both algorithms. The experiments also show that DERjoin is always advantageous when the duplicated values in relation R is 0.5 except the selectivity of relation R reaches to 0.1.

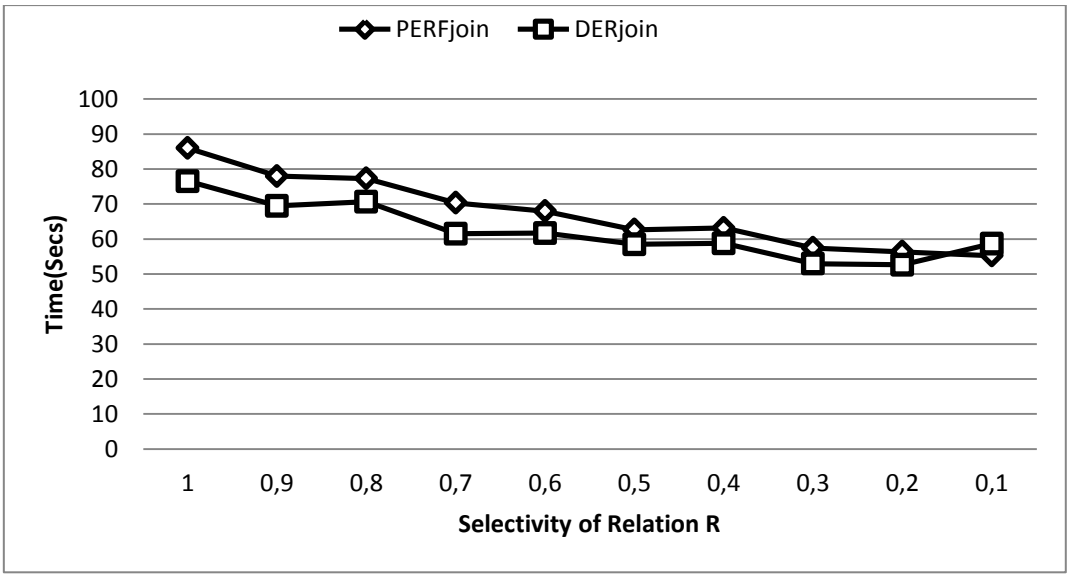

Figure 5.7. Execution time varying selectivity of relation R

In Figure 5.8 total bytes of data transferred between site 1 and site 2 is shown. It can be seen from the Figure that difference between PERFjoin and DERjoin does not change rapidly because the distinct join attribute values of relation R, are fixed to 5.000 at each test.

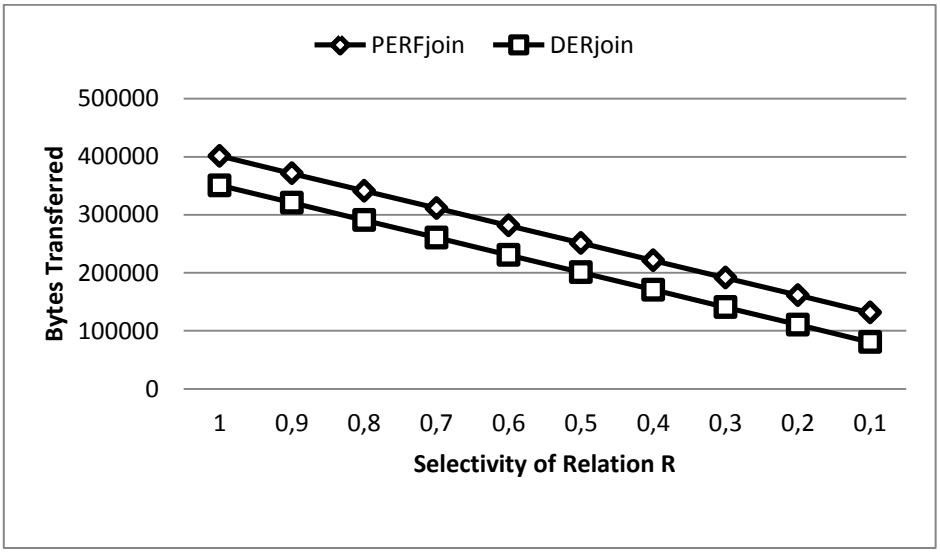

Figure 5.8. Total bytes of data transferred varying selectivity of relation R

5.6. Discussion of Results

In this performance test, execution time of join operation between relation R and relation S is measured by using DERjoin and PERFjoin with six different datasets. In the first experiment, the number of distinct join attributes of relation R is varied from 2.000 to 10.000 and it was showed that DERjoin operation is more advantageous if the duplicated values in relation R increase. DERjoin makes duplicate elimination before sending projected join attributes from site1 to site2 and if the rate of duplicated join attributes are high enough, then this makes a significant amount of reduction in total execution time of join operation between relation R and relation S.

In the second experiment the cardinality of relation R varies from 5.000 to 40.000 and percentage of unique join attribute values vary from 20% to 50%. Local processing cost of DERjoin operation increases when the cardinality of relation R increases. From this performance evaluation test it can be said that DERjoin operation is more advantageous than PERFjoin operation when the percentage of unique join attribute values and cardinality of relation R decrease.

In the third experiment the length of join attribute value is varies from 100bytes to 800bytes of data while selectivity is fixed to 0.5 and distinct join attribute values at relation R is fixed to 5.000. This experiment is performed for showing the relationship between size of data transferred and the time it takes to transfer it. It can be seen from Figure 5.6 that if the size of data increases then the transmission time increases and also DERjoin becomes advantageous while the size of data transferred increases.

In the forth experiment selectivity of relation R varies from 0.1 to 1 while distinct join attribute values are fixed to 5.000 and the size of join attribute value is fixed to 10 bytes. It can be seen from Figure 5.7 that while the selectivity of relation R decreases the time needed to process PERjoin operation comes close to DERjoin operation and also PERFjoin operation executes the join operation between relation R and relation S faster than DERjoin operation when the selectivity is 0.1. DERjoin operation adds some extra local processing cost while performing join operation between relation R and relation S and because of that reason when the selectivity is low, communication cost is minimum and the local processing cost shadows communication cost.

DERjoin is more advantageous when the rate of distinct join attributes in relation R decreases because it eliminates the redundant tuples before sending them to site2. When the rate of distinct join attributes is high, then PERFjoin becomes more advantageous because DERjoin adds some extra local processing cost to eliminate the duplicated values.

It is important to note that the experiments are performed in high speed bandwidth connection. However it is known that in real life when databases are connected to each other by internet it is mostly not possible to have a high speed bandwidth between databases. The experiments are performed in local area network connection because in internet connection it is not possible to connect computers to each other with a fixed bandwidth rate. If the bandwidth rate was low enough at those four experiments then it would be seen that DERjoin would be more advantageous when there was small rate of change in duplicated join attributes of relation R.

CHAPTER 6

CONCLUSION

Distributed database system consists of physically separated databases which are connected to each other with a communication network. Nowadays using distributed database and Client/server applications is very popular because the business environment needs reliable, accessible and scalable data. Distributing data among databases is advantageous than centralizing the data in one database. The distributed database system makes the information reliable, accessible and scalable.

A query in distributed database system can be processed with many different query processing strategies and query optimization is to find an efficient way of processing the query with the minimum cost among all query processing strategies. The cost of processing a distributed query is composed of local processing cost and transmission cost. Local processing cost is composed of CPU cost and I/O cost. Transmission cost is the cost of transmitting data from one site to another. In distributed query processing it is often needed to transfer data from one site to another and if the communication cost between sites is low then the communication cost may shadow the local processing cost. Most of the distributedquery processing algorithms are focus on reducing the transmission cost rather reducing the local processing cost.

Join operation is for combines different relations by using common attribute values. While performing join operation in distributed database system, if relations that participate in the join operation are located at different sites than they need to be transferred to the querying site and after the querying site receives the relations it makes the final join operation. In order to reduce the communication cost before sending them to the querying site, redundant data elimination and data compression can reduce the size of the data transferred. Redundant data consists of the tuples in a relation that are not going to participate in the final join operation result or the duplicated records.

Semijoin operation is a popular operation for reducing the volume of data transmitted between sites and there are many semijoin based previous works. In semijoin operation a small piece of information is exchanged between sites to give knowledge to the sites which tuples of relations are going to participate in the final join

operation result and this small piece of information is the projection of join attributes. Bloomjoin operation, which is an extension of semijoin operation, puts the projection of join attributes in a bit vector by using hash functions before sending them and the receiving site decodes the bit vector by using the same hash functions. Bit vector consists of 1s and 0s and size of a record in bit vector is just 1 bit. When the size of the bit vector is compared with the actual records, the size of a bit vector can be smaller than the actual records [25]. Using bit vector is a way of compressing the data, however using hash functions to encode and decode records might result in data loss because of the nature of hash functions. PERFjoin operation uses bit vector to reduce the communication cost as in bloomjoin operation, but it does not use any hash functions to prevent data loss while encoding and decoding values. Each bit in the bit vector created by PERFjoin operation gives information foreach tuple of therelationwhether they participate in the final join operation result or not. It is not possible to eliminate duplicated join attribute values before sending them in PERFjoin operation.

This thesis pointed out the challenges of processing join operation in distributed system in which sites are geographically separated and connected with low bandwidth. Reducing communication cost, preventing data loss and duplicated value elimination are challenges of the distributed join operation processing. To address these problems, a novel distributed join operation processing algorithm called Distinct Encoded Records Join (DERjoin) is proposed.

DERjoin is a semijoin based join algorithm, it consists of forward and backward phases. In the forward phase, as in semijoin operation distinct projection of join attributes are sent from one site to another. Then, the receiving site creates a bit vector which gives information about distinct projection of join attributes whether they participate in the final join operation result or not. In the backward phase the created bit vector is sent to the other site. It is not possible directly to eliminate redundantdata from relation by using the bit vector because the relation may contain duplicated values and if so the length of the bit vector is not going to be same with the cardinality of the relation, its length is going to be same with the distinct projection of join attributes. After the site receives the bit vector another bit vector that gives information about which tuples of the relation are traversed is need to be created. DERjoin operation creates and traverses two different bit vectors and it is clear that it increases the local processing cost. However if the connection speed between sites is low enough than the local processing cost is going to become negligible.

In the performance evaluation studies DERjoin operation is compared with PERFjoin operation. The reasons for choosing the PERFjoin operationfor comparison are: 1) It is a semijoin based join operation, 2) It has forward and backward phases 3) It uses bit vector for encoding the records. Performance evaluation studies show that if the rate of duplicated join attribute values is higher than performing join operation by using DERjoin takes less time, because the size of data transferred between sites can be reduced significantly.

In a semijoin based distributed join operation a part of a relation need to be sent to the other site to eliminate redundant tuples from both relations. The part of the relation is the projection of join attributes of one relation and it may contain high volume of duplicated records. If the rate of the duplicated values in the projection of join attributes is high then sending the projection of join attributes without duplicate elimination increases the transmission cost. Also rather than sending the actual records, compressing them beforesent significantly reduces the transmission cost.

In future, a comprehensive research is suggested to extend the DERjoin algorithm to make it possible to analyze the data. If it can be possible to analyze the data then the rate of duplicated values of join attributes can be measured and the algorithm can dynamically decide whether to make duplicate elimination or not. Another suggestion for further study is that by using real world environment the performance evaluation study should be constructed.

REFERENCES

1. Bealor T., "Semi-Join Strategies For Total Cost Minimization in Distributed Query Processing", Master Thesis, University of Windsor, Canada, 1995.

2. Donald Kossmann, **"**The state of the art in distributed query processing",ACM Computing Surveys,Volume 32 Issue 4, Dec. 2000.

3. Ramzi A. Haraty, and Roula C. Fany,"Query Acceleration in Distributed Database Systems",inRevistaColombiana de Computacion, Vol. 2, Nr. 1 , p. 19-34, 2001.

4. Apers,P.,Hevner,A.,Yao,A. "Optimization Algorithms For Distributed Queries", in IEEE Transactions on Software Engineering, Vol.Se-9,No.1 . pp. 57-68, 1983.

5. William Perrizo, Prabhu Ram, David Wenberg, "Distributed Join Processing Performance Evaluation", HICSS(2), pp. 236-245, 1994.

6. M.Tamer Özsu,PatrickValduriez, "Principles of Distributed Database Systems, Third Edition", 2011.

7. Alan R.Hevner,S. Bing Yao," Query Processing in Distributed Database Systems", IEEE Transactions on Software Engineering, 1979.

8. Fan Yuanyuan, Mi Xifeng,"Distributed Database System Query Optimization Algorithm Research",IEEE, 2010.

9. C.T. Yu,C.C. Chang,"Distributed Query Processing" ACM Computing Surveys, 1984.

10. Raef Abdallah,"Introducing Perf to a Query Optimization Algorithm", Master Thesis, Labanese American University, Lebanon, 1997.

11. Jo-Mei Chang, "A Heuristic Approach to Distributed Query Processing", Proceedings of the 8th International Conference on Very Large Data Bases, 1982.

12. Philip A. Bernstein ,Nathan Goodman ,Eugene Wong ,Christopher L. Reeve ,James B. Rothnie,"Query Processing in a System for Distributed Database (SDD-1)", ACM Transactions on Database Systems, 1981.

13. RiadMokadem,AbdelkaderHameurlain,Franck Morvan, "PerformanceImproving of Semi-join Based Join Operation through Algebraic Signatures", ISPA '08

Proceedings of the 2008 IEEE International Symposium on Parallel and Distributed Processing with Applications,2008.

14. Bernstein, P.A. and Chiu, D.W.,"Using Semi-joins to Solve Relational Queries", J.ACM 28, Jan.1981.

15. Burton H.Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors", July 1970.

16. Z. Li and K. A. Ross. Perf join: "An Alternative to Two-Way Semi-join and Bloomjoin",in Proceedings of the International Conference on Information and Knowledge Management (CIKM), pages 137–144, 1995.

17. S. Y. Sung, Peng Sun, Zhao Li, C. L. Tan, "Virtual-join: a Query Execution Technique", PCC '02 Proceedings of the Performance, Computing, and Communications Conference, 2002.

18. Hyunchul Kang Nick Roussopoulos, "Using Two Way Semi-joins in Distributed Query Processing", Proceedings of the Third International Conference on Data Engineering, 1987.

19. Priti Mishra Margaret H. Eich, "Join Processing in Relational Databases", ACM Computing Surveys(CSUR), Volume 24 Issue 1, March 1992.

20. Graefe, G., "Query Evaluation Techiniques for Large Databases",ACM Computing Surveys,Vol.25.No2,pp.73-169, 1993.

21. Ullman,J.D.,"A First Course in Database Systems",Prentice Hall, Third Edition, October 2007.

22. Ullman,J.D.,"Principles of Database & Knowledge Systems",Second Edition,Freeman,Vol.1,W.H.Company, March 1988.

23. Ramez Elmasri,ShamkantNavethe, "Fundementals of Database Systems",6th Edition, Pearson/Addison Wesley, 2010.

24. N. Roussopoulos, H. Kang"A pipeline- n way join algorithm based on 2-way Semijoin program", IEEE Transactions on Knowledge and Data Engineering, 1991.

25. Zhe Li , Zhe Li , Kenneth A. Ross , Kenneth A. Ross,"BetterSemijoins Using Tuple Bit-Vectors",Techincal Report CUCS-10-94,Colombia University, New York, 1994.

APPENDIX A

IMPLEMENTATION OF JOIN OPERATION

A.1. Codes for Communication between Site1 and Site2

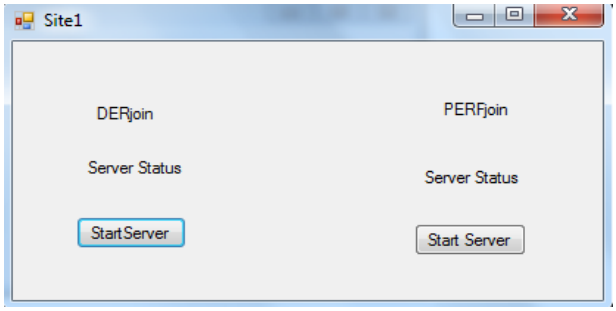

Figure A.1. Interface of application on computer1

Site 1 actually computer1 opens its port number 9995 for tcp communication manually by pressing "StartServer" for either DERjoin or PERFjoin operation. After site1 opens its specific port and starts listening than site2 can reach the methods in site1.

```
TcpChannel tcp = newTcpChannel(9995);
ChannelServices.RegisterChannel(tcp);
RemotingConfiguration.RegisterWellKnownServiceType(typeof(Methods),     "IJoin",
WellKnownObjectMode.SingleCall);
```

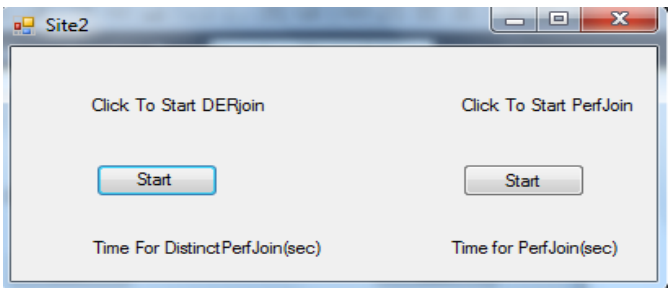

Figure A.2. Interface of application on computer2

After site1 opens its port for DERjoin or PERFjoin operation site2 can create a remote object as shown below;

```
publicvoid ConnectToSite1ForDistinctEncodedRecords() {
TcpChannel tcpChannel = newTcpChannel();
ChannelServices.RegisterChannel(tcpChannel);
Type requiredType = typeof(Site1Op.Methods);
            RemoteObject = (Site1Op.Methods)Activator.GetObject(requiredType,
"tcp://localhost:9995/Site1");


        }

privatevoid ConnectToSite1ForPerJoin(object sender, EventArgs e)
        {
            Site2Op.Methods methods = new Site2Op.Methods();
            methods.ConnectToSite1ForPERFjoin();
            methods.PERFjoin();
            time.Text = methods.elapsed.TotalSeconds.ToString();

        }
```

A.2. DERjoin and PERFjoin Operation Codes

The join operation accepts two parameters tableR.txt and tableS.txt and it produces final_result.txt as the result at site2. The codes for DERjoinoperation performed at site2 is shown below;

```
publicvoid DistinctEncodedRecordsJoin()
        {
DateTime start = DateTime.Now;
            Convert("c:\\veri2.txt", "myTable2", ",");
            RemoteObject.Get_Distinct_Project();
DataTable Projection = RemoteObject.Retun_Projection();
DataTable reducedTable2 = SemijoinOperation(Projection, data.Tables[0]);
bool[] bitVector=CreateBitVector(reducedTable2, Projection);
DataTable table1= RemoteObject.ReduceWithBitVectorNew(bitVector);
            LastJoin(table1,reducedTable2);
            elapsed = DateTime.Now - start;
        }
```

The object RemoteObject is for accessing the methods at site1 and those methods are Get_Distinct_Project() and ReduceWithBitVectorDist(bitVector). Get_Distinct_Project() is for taking distinct projection of relation R, and it is shown below;

```
publicvoid Get_Distinct_Project()
         {
             Convert("c:\\veri1.txt", "myTable", ",");
             Projected_Values = newDataTable();
             Projected_Values.Columns.Add("id");
foreach (DataRow row in data.Tables[0].Rows)
             {
bool contains = false;
foreach (DataRow row2 in Projected_Values.Rows)
                 {
```

```
if (row["id"].ToString() == row2["id"].ToString())
                        {
                            contains = true;
                        }

                    }
if (!contains)
                    {
                        Projected_Values.Rows.Add(row["id"]);
                }

            }


        }
publicDataTable Retun_Projection() {
return Projected_Values;
        }
```

The method ReduceWithBitVectorDist() takes the bit vector created by site2 as parameter and reduces the relation R by using the bit vector as shown below;

```
publicDataTable ReduceWithBitVectorDist(bool[] bitVector)
         {
int sizeOfTable = data.Tables[0].Rows.Count;
bool[] CreatedBitVector=newbool[sizeOfTable];
DataRow tempRow;
DataTable projectedValues=newDataTable();
             projectedValues.Columns.Add("id");
projectedValues.Columns.Add("name");
tempRow = data.Tables[0].Rows[0];
             Convert("c:\\veri1.txt", "myTable", ",");
for (int i = 0; i < bitVector.Length; i++)
             {
for (int j = 0; j < sizeOfTable; j++)
                    {
                      if(tempRow["id"].ToString()==data.Tables[0].Rows[j][
                      "id"].ToString())
                        {
                            CreatedBitVector[j] = true;
if (bitVector[i] == true)
                                    {

                            projectedValues.Rows.Add(data.Tables[0].Rows[j
                            ]["id"], data.Tables[0].Rows[j]["name"]);
                                    }
                        }

                    }
for (int t = 1; t < sizeOfTable; t++)
                    {
if (CreatedBitVector[t] == false)
                        {
                            tempRow = data.Tables[0].Rows[t];
break;
                        }

                    }
             }
return projectedValues;
         }
```

The codes for PERFjoin operation is shown below, the object RemoteObject is again for accessing the methods at site1 and those methods are Return_Projection() and ReduceWithBitVectorDist(bitVector).

```
publicvoidPERFjoin() {

DateTime start = DateTime.Now;
            Convert("c:\\veri2.txt", "myTable2", ",");

             RemoteObject.Get_Project();
DataTable Projection = RemoteObject.Return_Projection();
DataTable reducedTable2 = SemijoinOperation(Projection, data.Tables[0]);


bool[] bitVector = CreateBitVector(reducedTable2, Projection);
DataTable ReducedTable= RemoteObject.ReduceWithBitVector(bitVector);
            LastJoin(ReducedTable,reducedTable2);
            elapsed = DateTime.Now - start;

 }
```

The method Return_Projection() is for taking the projection of join attributes of relation R as shown below;

```
publicvoid Return_Projection()
             {
                 Convert("c:\\veri1.txt", "myTable", ",");
                 Projected_Values = newDataTable();
                 Projected_Values.Columns.Add("id");
foreach (DataRow row in data.Tables[0].Rows)
                 {
                     Projected_Values.Rows.Add(row["id"]);

                 }


                 return projection;

             }
```

The method ReduceWithBitVectorDist() takes the bit vector created by site2 as parameter and reduces the relation R by using the bit vector as shown below;

```
publicDataTable ReduceWithBitVector(bool[] bitVector)
         {
             Convert("c:\\veri1.txt", "myTable", ",");
int size = bitVector.Length;
DataRow row;
DataTable tableReduced=newDataTable();
             tableReduced.Columns.Add("id");
             tableReduced.Columns.Add("name");
for (int i = 0; i < size; i++)
             {
if (bitVector[i] == false)
             {
```

```
tableReduced.Rows.Add(data.Tables[0].Rows[i]["id"],data.Tables[0].Rows
[i]["name"]);

                }

            }
return data.Tables[0];
        }
```

APPENDIX B

GENERATED DATA FOR JOIN OPERATIONS

There are two datasets generated for the join operation named relationR.txt and relationS.txt. RelationR.txt is stored at computer1 and RelationS.txt is stored at computer2. Relation R has two field names attribute1 and attribute2 and relation S two field names attribute1 and attribute3 and attribute1 is the common join attribute value. The data is generated using an open source program Spawner.

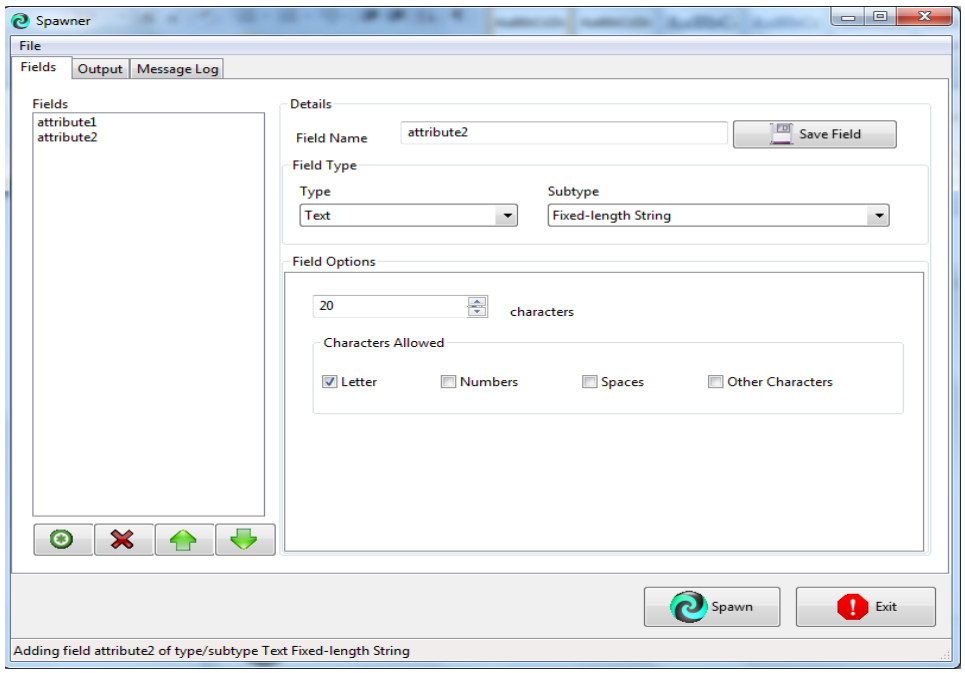

Figure B.1. Interface of Spawner

It is possible to create txt file by using Spawner. Also Spawner makes it possible to specify how many characters is going to be placed in a field, which characters are allowed and number of records that is going to be generated can be specified. After those specifications are given Spawner generates txt file which is filled by the randomly generated records.

After data is generated by using Spawner, selectivity and the number of the duplicated join attributes of relation R and relation S are generated by using C#.NET Windows Form Application.