Sub Query Site Allocation and Optimization for a Distributed Query

A Dissertation submitted

By

PARMINDER KAUR

to

Department of Computer Science and Engineering

In partial fulfilment of the Requirement for the

Award of the Degree of

Master of Technology in Computer Science and Engineering

Under the guidance of
Mr. Robin Prakash Mathur

(Assistant professor)

(April 2015)

LOVELY
PROFESSIONAL
UNIVERSITY

School of: Computer Science & Engineering (CSE)

DISSERTATION TOPIC APPROVAL PERFORMA

Name of the Student: Parminder Kaur
Registration No: 11300122
Batch: 2013-2015
Roll No: RK2306A05
Session: 2014-15
Parent Section: RK2306A

Details of Supervisor:
Designation: Asst. Professor
Name: Robin P Mathur
Qualification: M.Tech-CSE
UID: 14597
Research Experience: 4 Yrs

SPECIALIZATION AREA: Database System (pick from list of provided specialization areas by

PROPOSED TOPICS

Query Optimization in Distributed Database System
Fragmentation/Partitioning in Distributed Data
Database Integrity in Distributed database Sy

Signature o

PAC Remarks:

First topic approved

11179

11041 30/9/14

Signature:

APPROVAL OF PAC CHAIRPERSON:

Supervisor should finally encircle one topic out of three proposed topics and put up for approval befo
Committee (PAC)
Original copy of this format after PAC approval will be retained by the student and must be attached
Project/Dissertation final report.
One copy to be submitted to Supervisor.

ABSTRACT

Distributed databases are the outcomes of top notch technology advances, high speed computer networks further facilitated its growth and its suitability in satisfying various businesses needs make it more popular. As data resides at different sites in a distributed database environment, so to acquire a specific type of data; subdivision of a query into its sub-parts (sub-queries) is required and those sub-queries needs to be executed at different data sites. In some cases combination of data from two or more different sites may be required. To attain this goal a join operator is used. But using join is not always advantageous in terms of cost as it may sometimes result in more communication cost in cases when complete relation is not desired for join operation. In such scenario communication cost involved between two sites can be reduced using other forms of joins like inner join. Inner join is also not always useful. So a need of finding the appropriate strategy to decide and assign join operations arises. In this thesis join operator allocation has been done dynamically by dynamically calculating percentage participations for joins and inner joins for the dynamic distributed database simulated. This dynamic percentage participation is given as input to the simulator built in MATLAB based on which fragment size for join operation is calculated. The simulator by using the genetic algorithm computes the minimum communication cost involved in executing the query under different cases using joins only, using inner joins only, and mixture of both. Hence finding the optimal query design for a distributed database using mix of joins is attained.

CERTIFICATE

This is to certify that Parminder Kaur has completed M.Tech dissertation titled **Sub Query Site Allocation and Optimization for a Distributed Query** under my guidance and supervision. To the best of my knowledge, the present work is the result of her original investigation and study. No part of the dissertation has ever been submitted for any other degree or diploma.

The dissertation is fit for the submission and the partial fulfilment of the conditions for the award of M.Tech Computer Science & Engg

Date: 24 April,2015

Signature of Advisor

Name: Robin Prakash Mathur

(Asst. Professor, School of Computer Science & Engineering)

Lovely Professional University

Phagwara, Punjab (144402).

ACKNOWLEDGEMENT

I would like to thank LOVELY PROFESSIONAL UNIVERSITY for giving me opportunity to use their resource and work in such a challenging environment. I am grateful to the individuals whom contributed their valuable time towards my thesis.

I wish to express my sincere and heart full gratitude to my guide “Mr. Robin Prakash Mathur” Assistant professor, who guides me to take up this thesis in **Sub Query Site Allocation and Optimization for a Distributed Query**.

I would also like to offer my sincere and heartfelt thanks to Prof. Dr. R.S. Virk of Computer Science And Engineering Department, G.N.D.U., Amritsar for his continuous guidance and support throughout my dissertation.

Without his unconditional help I would never have been able to complete my work. I would also like to extend my gratitude to my friends and family who always encouraged and supported me in this thesis work.

Last but not the least; I would like to thank all the staff members of department of Computer Science & Technology who have been very patient and co-operative with us.

DECLARATION

I hereby declare that the dissertation entitled, **Sub Query Site Allocation and Optimization for a Distributed Query** submitted for the M.Tech Degree is entirely my original work and all ideas and references have been duly acknowledged. It does not contain any work for the award of any other degree or diploma.

Date: 24 April,2015

Investigator
Parminder Kaur
Regn. No. 11300122

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In the old days, programs stored data in the regular files. Each program has to maintain its own data which results in huge overhead and easily prone to error. The development of database management helped to fully achieve data independency that provides centralized and controlled data maintenance and access. Also application is immune to physical and logical organization. The advancement in database and communication technologies enhanced the popularity of distributed databases, as it provides high availability, autonomy, and affordability for managing large databases. A distributed database can be considered as a collection of data which are distributed over different sites of a computer network. Each site of the network is capable to perform local applications autonomously. However the distributed database systems are used in applications which require access to an integrated database from geographically dispersed locations. The location of data items and the degree of autonomy of individual sites play a prominent role in all aspects of the system. Data allocation is the prominent activity in the distributed database which decides that where to locate the data. [4], [5]. Data is the base of whole world of growing organizations in today's world and managing data is one of the most trivial tasks. Database Management System are used to manage whole data in organizations. In today's world of universal dependence on information systems, every user of the system whether an employee or a employee need access to company's databases. Database is managed using two approaches known as Centralized Database Management System and Distributed Database Management System [8]. Conventionally, databases of any organization were focused at one mainframe location with all over wide-reaching access. Unified system management and could be beneficial when manager in a structured style but it posed few glitches as well. Thus, substitute strategy to the centralized database is distributed database. Distributed database is a collection of logically interrelated databases that can be stored at different computer network sites. The objective of a distributed database management system (DDBMS) is to control the management of a distributed database (DDB) in such a way that it appears to the user as a centralized database.

1.1 DATABASE MANAGEMENT SYSTEM (DBMS):

Database management system (DBMS) [8] is software [9] systems, which are basically collection of interrelated data and allow definition, creation, updating of databases [24]. DBMS are applications that are designed in such a way that they can interact with users, other applications, and database itself to analyze and capture data. DBMS provide a lot of facilities, some of which are:

1.1.1 Data Definition Language [31]: DBMS provide its users facility to define database, using data definition language (DDL). Users can specify structure of database, data types and constraints on data by using DDL.

1.1.2 Data Manipulation Language [19]: DBMS provide its users facility to insert, retrieve, update and delete data by using Data Manipulation Language (DML). DML provide general facility to enquire about data, which is known as query language.

1.1.3 View Mechanism [6]: DDL is also use to define a view. A view is basically a subset of database, but it doesn't form part of physical schema. Each user can have his or her view of database.

A distributed database (DDBMS) is such a database system in which all storage devices are not all attached to a CPU (central processing unit) and all of these storage devices are managed by distributed database management system.

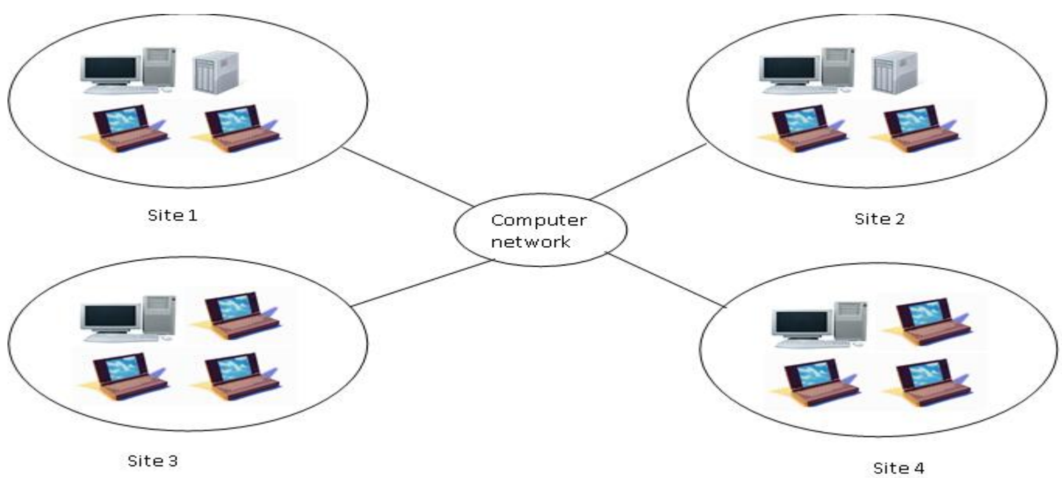


Figure 1: Distributed Database System (4 sites)

Such a database can reside in same large room, but all the fragments/replicas (stored at different sites) communicate with each other through network instead of shared memory.

1.2 BENEFITS OF DISTRIBUTED DATABASE [22]

i. **It reflects organizational structure**

A number of organizations in the world are distributed over several locations. For example: A bank has many offices in different cities of same district. The database used for such applications is distributed over many locations. Banks may keep database at each department containing details of employees, staff related to that department.

ii. **Improved share ability and local autonomy**

By distributing the database, data can be placed at site near to the use who frequently use that data. By doing so, locality of reference gets improved. Along with that, users get local control over data and can enforce policies regarding use of data. A global database administrator manages the entire global database, while duties of managing local database can be assigned to local administrator.

iii. **Improved Availability**

While using centralized database**,** failure of the central application can result in failure and unavailability of the whole system. But in case of distributed system, failure of one site, does not make whole system unavailable. Distributed databases are designed in such a way that they continue to function even in case of failure of one or two sites.

iv. **Improved Reliability**

Allocating data at different sites and maintaining replicas of data at various sites, the failure of a site does not make data inaccessible.

v. **Improved Performance**

The data, in case of distributed database is located at the site or near the site which most frequently accesses that data, because of which the speed of data access increases, communication cost incurred while accessing data from remote site gets reduced and hence, the performance of the system improves.

vi. **Modular Growth**

In case of distributed databases, expanding the system becomes easy. New sites can be added to the system at any time without affecting the whole system.

1.3 Distributed Database Design

Design [7] of a distributed database system is one of the most crucial aspect behind the success or failure of such a system. Designing a distributed system involves taking decisions related to the placement of data and programs in system (includes network nodes and network design itself). While designing a distributed system, the main focus is given to the division and placement of data i.e. to the placement of data. The issues that arises while designing a distributed database system are:

a. Why fragment at all
b. How to fragment
c. How much to fragment
d. How to test correctness
e. How to allocate

Two basic strategies used in designing distributed database system are:

- Top-down Approach
  - Involves designing the system from scratch
  - Used for homogeneous systems.
- Bottom-up Approach
  - Used for systems where database already exists at some sites
  - The aim is to connect the databases to solve common tasks.

1.3.1 TOP DOWN APPROACH TO DESIGN [27]

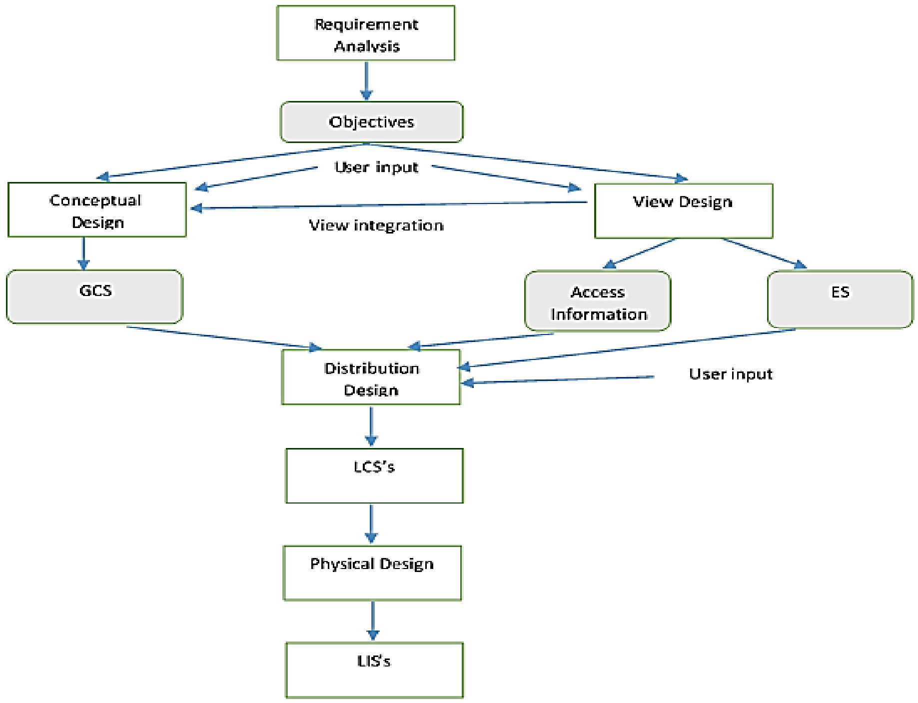


Figure 2: Top Down Approach to Distributed Database Design [4]

The top down approach to distributed design starts with requirement analysis of actual environment. It involves designing a system from scratch. This process involves creating data models which defines high level entities of the system and their relationships. Then refinements are applied to the high level data models to identify and define corresponding low level entities, their relationships and attributes.

The steps involved in top down process are:

- Analyze the requirements
- View integration and conceptual design
- Data distribution design
- Local physical schema design

- The process of designing starts with analysis of the requirements which defines the system. The requirement document is essential requirement for two parallel activities: conceptual design and view design.
- View design involves defining user interface, while conceptual design includes examining the system to determine its component types of entities and relationship

between entities. Conceptual design can also be defined as integration of user views. This integration is very important because the conceptual schema or design must not only support existing application, but must also support global conceptual schema and information about access patterns.

- So, the objectives at this step is to design conceptual schemas using local distribution entities across nodes of the system. In relational model instead of whole relation their fragments are distributed across the system.
- So, distributed design activity consists of two steps: fragmentation and allocation.
- The last step includes physical design, which involves making connections between conceptual schemas and physical storage devices on the nodes of corresponding data.

1.3.2 BOTTOM UP DESIGN APPROACH

Bottom up approach can be used for designing database of an existing system. Most of the times, existing and heterogeneous databases are integrated to a common distributed database system. This approach comprises of integrating existing schemata into a single global schema. But the following aspects must have fulfilled:

- A common database model must be selected for describing the global schema.
- Each local schema must be translated into the common data model
- The integration of common schema into common global schema: the merging common data definitions and resolving conflicts among different representations given to the same data.

The bottom up approach involves solving these three problems. The design steps are just reverse of the bottom up approach. The steps of integration for designing a new system are:

- Common data model selection
- Translation of each local schema into common model
- Integration of the local schema into a common global schema
- Design the translation between the global and local schemes.

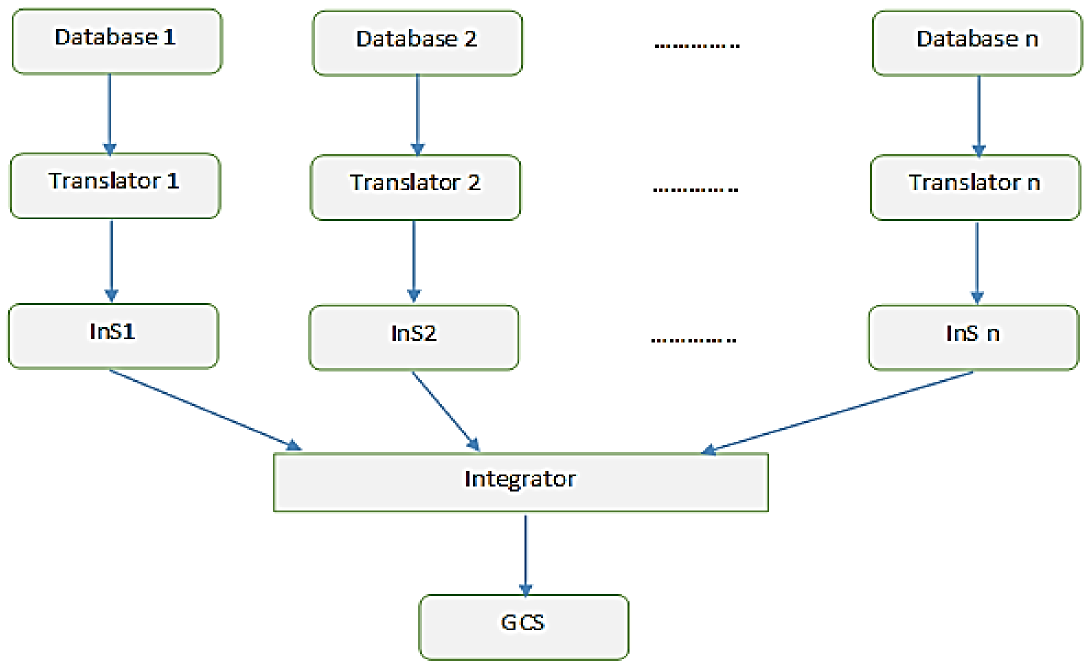


Figure 3: Bottom up Approach to Distributed Database Design

1.4 QUERY PROCESSING

Database Query is a way of instructing DBMS to update, insert, retrieve, and delete data from database. It can be defined as a request for information from database. The actual operation is performed by executing a number of low level operations. Such operations for example can be, select, project, join etc. Query Processing [5] is a process of transforming a high level query into correct and efficient execution plan, which is expressed by using a low level query and executing that query plan. All the activities involved in executing a query are included in it.

1.4.1 QUERY PROCESSING PROBLEM [13]

The main aim of query processor is to transform a high level query into low level query plan. Query processing becomes much more important in case of distributed databases. As in case of distributed systems, relations involved in query may be fragmented or replicated, and hence increase communication cost. In distributed databases, data is fragmented or replicated to increase locality of reference and parallel execution. So, the role of a distributed query processor can be defined as, mapping a query on distributed databases into a sequence of operations on fragments of relations. The data used by the low level query must be completely localized, so that operations bear on local fragments. The transformation must be correct and efficient.

Example: Consider the following query,

"Find name of all employees who manage a project",

On relational schema given as:

Table 1: Example of Database

emp:

ENO	ENAME	TITLE
E1	John	Electrical Engineer
E2	Steve	System Analyst,
E3	Bob	Mechanical Engineer
E4	Mary	Programmer
E5	Maria	System Analyst

responsibility:

ENO	JNO	RESP	DUR
E1	J01	Manager	12
E2	J01	Analyst	24
E2	J02	Analyst	6
E3	J03	Consultant	10
E5	J04	Manager	48

Relational calculus query equivalent to above query in SQL is**:**

Select ename from emp, responsibility where emp.eno= responsibility.eno and responsibility.resp="manager"

Two equivalent relational algebra queries for the above SQL query are:

Π_{ename} ($\sigma_{resp=\text{"manager" and emp.eno=responsibility.eno}}$ (emp × responsibility))

And

Π_{ename} (emp $\bowtie_{eno}$ ($\sigma_{resp=\text{"manager"}}$ responsibility))

From above example, it can be observed that:

I. First query uses Cartesian product, which uses more computing resources, so it must be avoided, while the second one uses join instead of Cartesian product, so it must be retained.

II. In case of a centralized system, the only task to be performed by query processor is to convert relational calculus query into the best relational algebra query (which uses minimum resources).

But in case of distributed system, execution plan is expressed by using relational algebra query along with communication operators for exchanging data between sites. The best site to process data must be selected along with the best ordering of relations and operators.

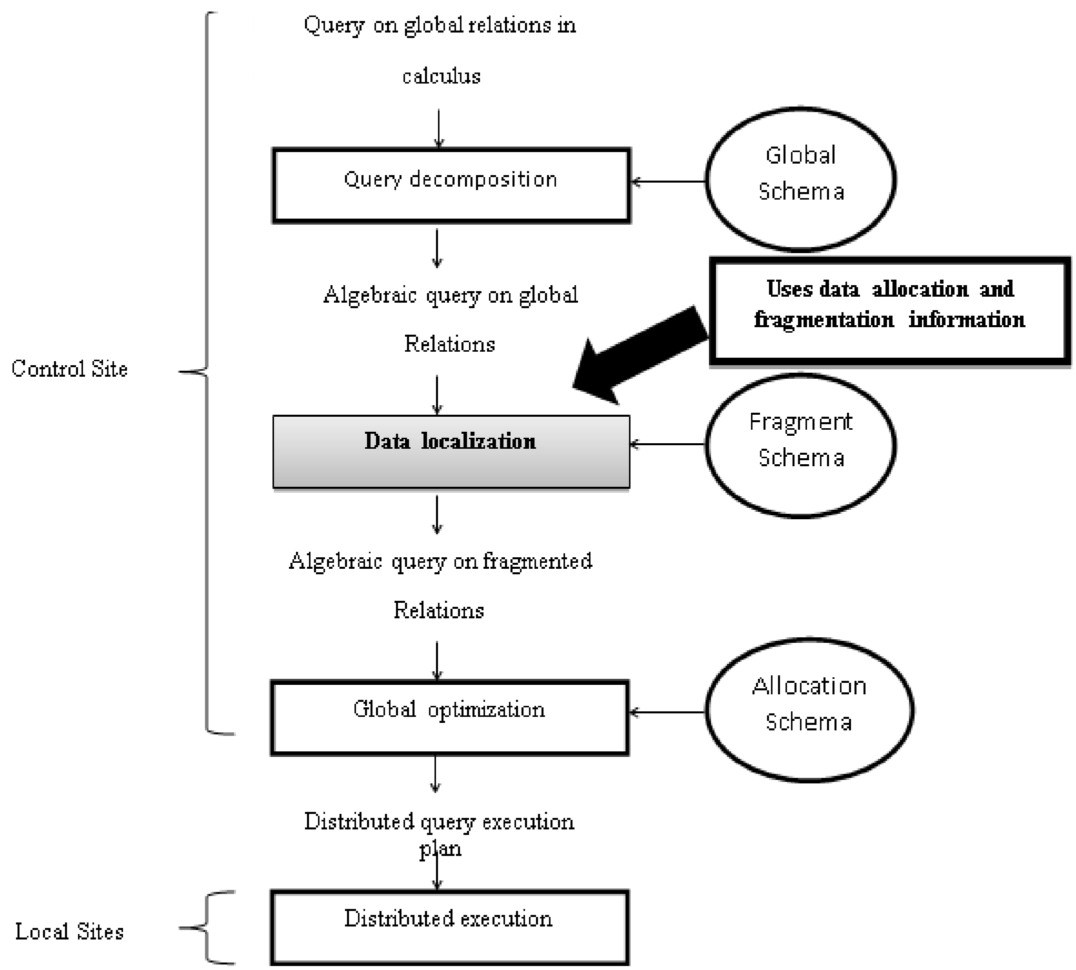
1.4.2 QUERY OPTIMIZATION[4]

As illustrated in above example, a query can be expressed by using two or more equivalent query plan. There can be a huge difference between costs of two alternative plans, depending upon the processing costs at different sites, communication costs etc. Query optimization [4] is the function of determining the most efficient query plan among all, which is performed by query optimizer.

1.4.3 LAYERS OF QUERY PROCESSING [28]

The problem of query processing has been subdivided into various layers each corresponding to various sub problems. The first three layers shown in the figure perform the task of mapping the input query to an optimized distributed query execution plan. They perform query decomposition, data localization and global optimization functions. A central control site performs functions of first three layers and uses global directory's schema information. The last layer executes the optimized query execution plan and returns the answer to that query.

I. **Query Decomposition [33]:** The first layer uses the information in global conceptual schema, which contains global relations. It decomposes the input calculus query into algebraic query which bears on global relations. It doesn't need any information about distribution of data on various sites. Hence, it uses techniques used in centralized system.



I. Figure 4 : Steps of Query Processing

Query decomposition is a four step procedure**.** The steps involved are:

1. Frist of all, query is normalized to a form which is best for further manipulation. Normalization is done by manipulating query quantifiers and by applying logical operator priority.
2. Next step is of sematic analyses, which attempts to detect and delete incorrect queries as early as possible.

3. Then the query is simplified, which involves eliminating redundant predicates.
4. At last the query is restructured as algebraic query.

II. **Data Localization:** the input to this layer is algebraic query produced in the first layer. This layer uses the data distribution information in fragment schema to localize the query's data. The main role of this layer is to determine all the fragments involved in query and then transform the query on global relations into a query which bears on fragments.

III. **Global query Optimization:** it takes the query on fragments as input. The main goal of this layer is to determine the optimal execution plan for the query. An execution plan is described by using relational algebra operators and communication primitives. This layer finds out the best execution plan which involves best ordering of algebra operators and communication operators, which minimizes the total cost of executing the query. The cost function is described in terms of time units.

IV. **Distributed query execution:** All sites containing the fragments involved in query perform the task of last layer. Each sub plan (sub query) is executed at one site, called local query and then is optimized using local schema. Algorithms of centralized systems can be used in local optimization.

1.4.4 OBJECTIVES OF QUERY PROCESSING [25]

- The main objective of query processor is to convert a high level query on a distributed environment, which appears a single database to users, into an efficient execution strategy in a low level language on local databases.
- An important aspect of query processing is query optimization. There may be more than one execution strategies which are correct transformation of high level query, the one that optimizes the resource consumption must be retained.
- The good indicators of optimized resource consumption are:

- The *total cost* incurred while processing the query. It is sum of all cost incurred while processing the operations of the query like, input-output cost, communication costs (in case of distributed systems), CPU costs.
- The *response time* of query. It is the time taken by system in executing the query. Because of parallel execution at one or more sites, response time of a query may be less than its total cost.

- Minimizing the total cost is one of the main objectives of query processing.

1.5 DATA ALLOCATION AND FRAGMENTATION

While designing a distributed database system, the major issues involves distributing the central database to various sites. This involves fragmenting the database and allocating fragments to various sites. The design of system must be based on both the quantitative and qualitative information.

FRAGMENTATION: The process of dividing the relation into sub relation is called fragmentation. These fragments are then distributed to different sites. Defining and allocating the fragments to sites must be based on the access pattern of different applications. Two fundamental strategies for fragmentation are:

a. Horizontal Fragmentation [3]: It partitions a relation along its tuples. So, a fragment is basically a subset of tuples of relations.

b. Vertical Fragmentation: Vertical fragmentation partitions a relation long its attributes i.e. fragments of a relation in vertical fragmentation produces fragments, each of which contains subset of attributes of main relation.

ALLOCATION [16]: One major task while designing a DDBMS is to allocate resources to various computer nodes or sites. Data or fragment allocation must be done in such a way that locality of reference gets maximized. Four general strategies for data allocation are:

I. Centralized
II. Partitioned
III. Complete replication
IV. Selective Replication

1.6 JOINS

The joins impact can be seen when a client prerequisite join datasets from a relational database[2]. The space has turned into a colossal. Associations gathering information because of expanding rate swinging to this information to drive their objectives. Disadvantage of gathering this information is the need to some way or another capably store and structure it. One of the key inhabitants of db configuration is to convey it to condition of standardization. It is accomplished by which imitation information is lessened, capacity is enhanced and the need to get to this standardized information that has been put away in partitioned tables gets to be vital. A join in its pith will return, erase, or redesign information from more than one information source as one arrangement of information.

1.6.1 Cartesian join

The Cross join is the basic among rest of joins to compose and perceive. It is basically a gathering of two relations without a qualifying Where proviso. The result of this join is each tuple of the 1st connection consolidated to each tuple of the 2nd connection.

1.6.2 Equijoin

An Equijoin is the one of the straightforward kind of inward join and can be recognized by the equivalent sign predicate between two connecting variables in the Where condition.

1.6.3 Outer joins

This joins are just ready to process two tables at once. There are three principle sorts of external joins: Left, Right, and Full. Since an Outer join can just join two tables at once consider the first table as the Left hand side table and the second table as the Right Hand side table, in this manner issuing us the Left Outer Join and the Right Outer Join.

1.6.4 Left Outer Join

On account of a Left Join the table that goes before the essential word "Left Join" in the from proviso is viewed as the Master information set. That implies all lines paying little respect to whether they have a coordinating column on the Right hand side table will be kept in the last information set. For the situation where there are various columns in the Right hand side table, the Left hand side table's information will be copied.

Objective of join in distributed databases

Join query execution is more complex in a distributed database than in a centralized database. In a distributed database, to join two files that are located at different sites, data from one of the files must be transmitted to the site of the other file (or data from both files must be transmitted to a third site). This data transmission could be time-consuming if data transmission data is more. Therefore, distributed database systems need to transfer the data as fast as possible in order to improve join query performance. There are two basic join query execution methods used in the distributed database systems [26].

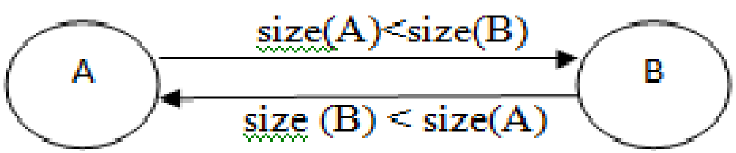


Figure 5: Transfer of operands

One method is to transfer the smaller table of two join query participating tables. This method can efficiently perform the join query which the quantity of result is much less than the quantity of two source tables. Another way is to transfer two tables in parallel. Parallel transmission can reduce the response time for the join query which the quantity of result is equal to or greater than the quantity of two source tables. The main objective of join query optimization is to reduce the cost of data transmission, small volume of transmitted data and move data in parallel so as to minimize the response time. Therefore, these two methods are not good for all types of join query.

To minimize the amount of data transmission between the sites inner join operator is used. Using Join, entire data most of the tuples in relation participate in the join but in case of inner join the size of relations are reduced [26].

Inner joins

In order to join two sub queries involving data from multiple sites using join query, data has to be transmitted from one site to other. This transmission of data increases the communication time. So the optimizer must consider efficient order in which tables are joined in such a way that communication overhead has cut down.

There is a problem of finding an efficient join order for a query because query Optimizer has to examine number of existing substitutions. Also, join operation affects the size of result of particular fragment to increase or decrease. The estimation of join results is quite difficult. Join query execution is time-consuming and more complicated on a distributed database than on a traditional centralized database if those two tables participating in a join query are stored on different remote sites. An approach to implement this join query on a DDB is to send one of the join participating tables to the site of the other table and perform the join at that site. Join ordering in distributed queries is done by two approaches. First one is to optimize directly the ordering of join and another is to substitute join by groupings of inner joins to reduce communication cost [5]. It is very useful in improving a join by minimizing the data transferred. Join reducers were put in to reduce the communication costs of distributed database systems [6].

1.7 Genetic Algorithm

Genetic Algorithm (GA) is initially grown by John Holland, his partners and his understudies at college of Michigan in the 1960s and 1970s. Holland's objective was to study the wonder of adjustment as it happened in nature and to create courses in which the components of common adjustment may be imported into computer frameworks. Unique objective of the examination was to clarify the adaption of characteristic frameworks and to outline simulated frameworks that attempt to grasp versatile and vigorous properties of regular frameworks

[29]. Holland's Genetic Algorithm was a system for moving from one population of "chromosomes" to another population by utilizing a sort of "characteristic determination" together with the genetics inspired administrators of hybrid, change, and reversal. Every chromosome comprises of "qualities" (e.g., bits), every quality being an occasion of a specific "allele". The determination administrator picks those chromosomes in the population that will be permitted to imitate, and by and large the fitter chromosomes deliver more posterity than the less fit ones. Hybrid trades sub parts of two chromosomes, generally copying natural recombination between two single−chromosome ("haploid") creatures; transformation haphazardly changes the allele estimations of a few areas in the chromosome; and reversal inverts the request of a coterminous segment of the chromosome, in this manner revising the request in which qualities are displayed [30].

1.7.1 Working of Genetic Algorithm

Genetic algorithm starts working on a randomly generated set of solutions, known as initial population. Each solution is represented by a fixed length string of binary numbers (i.e. 101010…). Fitness is connected with every arrangement. The fitness assessment is depend on objective function. In this every string representing to the arrangement is called chromosome, every bit of the string is known as the gene. The arrangement of strings is called populace. The chromosomes advance through progressive cycles, called generations. Amid every era, the chromosomes are assessed utilizing some measure of fitness [33].

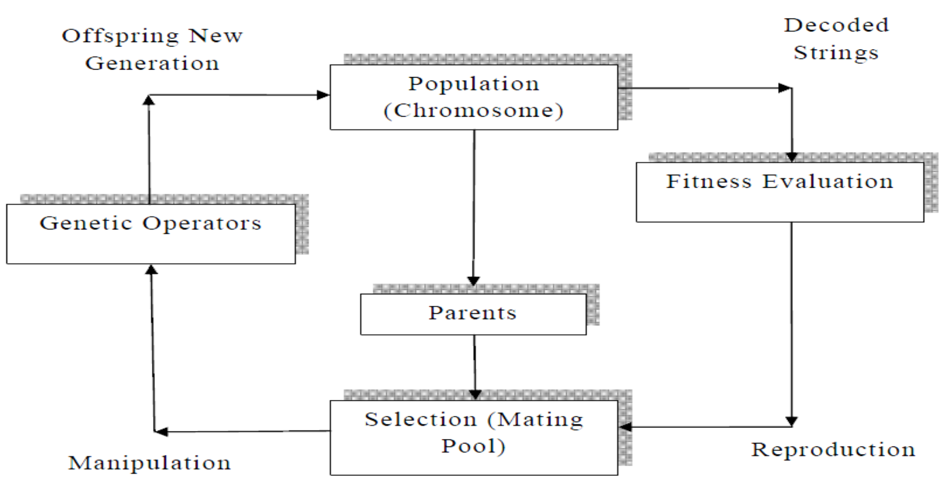


Figure 6: A flowchart of working of Genetic Algorithm [33].

To create the next generation, new chromosome called offspring, are formed by either:

- Merging two chromosomes from the parent generation using a crossover operator.
- Modifying a chromosome using a mutation operator.

A new generation is formed by:

- Selecting, according to the fitness values, some of parents and offspring.
- Rejecting others so as to keep the population size constant.

Fitter chromosome have higher probabilities of being chosen, after a few eras, the calculation meets to the best chromosome, which states to the ideal and suboptimal answer for the issue. Induction is thought to be arbitrary. Recombination regularly includes hybrid and transformation to yield offspring.

1.7.2 Outline of the Basic Genetic Algorithm

1. [Start] Chromosome of length n is produced from random population.

2. [Fitness] the fitness function of each chromosome is calculated from the population

3. [New population] New population is created by iterating following steps.

I. [Selection] Select two chromosomes are chosen from a population based on their fitness.

II. [Crossover] Cross over the parents with crossover probability.

III. [Mutation] with a mutation probability mutate new offspring at each locus (position in chromosome).

IV. [Accepting] New population is created by placing new offspring.

4. [Replace] new generated population is required for a further run of algorithm.

5. [Test] If the end condition is satisfied, stop, and return the best solution in current population

6. [Loop] Go to step 2

The process is iterated sequentially to produce new population. Process is iterated for criterion to met [33].

1.7.3 GA Operators

The GA includes three fundamental genetic operators: Reproduction, Crossover and Mutation. These operations are used to select and manipulate population solutions and select the most appropriate offspring to pass on to the succeeding generations [29].

1.7.4 Reproduction

Reproduction selects good strings from the population and puts them in mating pool [34]. The idea is to pick up the strings with higher fitness from current population and apply genetic operators to new strings for the successive population. The fittest chromosomes may be chosen a few times, be that as it may, the quantity of chromosome chose to replicate is equivalent to the populace size, in this way, keeping the size consistent for each era. This stage has a component of irregularity simply like the survival of life forms in nature. The most normally utilized choice techniques are taking after:

- Roulette Wheel Selection
- Stochastic universal sampling
- Ranked selection
- Truncation selection
- Tournament selection

The roulette wheel is probably the most popular technique used as the selection method for genetic algorithm. In this method, the entire population is represented by a segmented wheel [34]. The total number of segments in the wheel corresponds to the number of individuals in the population. Each individual is represented by a segment according to its fitness value. The more fit individuals will have bigger segment on the wheel and thus, will have better chances of passing their genes along to the next generation. On the other hand, poorly fitted individuals get less chances of passing the genes on to the next generation.

In the tournament selection method, n individuals are randomly selected from the population. The fit individual from this group will have its genes passed along the next generation via the crossover procedure. This procedure is repeated until enough individuals have been selected to reproduce and create the next generation.

1.7.5 Crossover

Crossover is a genetic operator that combines (mates) two chromosomes (parents) to produce a new chromosome (offspring). The thought behind hybrid is that the new chromosome may be superior to both of the folks in the event that it takes the best attributes from each of the folks. Hybrid haphazardly picks a locus and trades the sub arrangement previously, then after the fact that locus between two chromosomes to make two posterity, e.g.

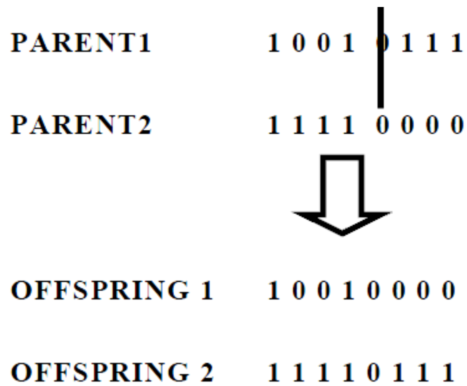


Figure 7: Crossover Operation in GA [34].

Commonly used combination techniques are as follows:

• One point crossover

• Two point crossover

• Uniform crossover

• Partially Matched Crossover (PMX)

• Order Crossover

• Cycle Crossover

1.7.6 Mutation

Mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state. This can bring about altogether new quality qualities being added to the quality pool. With these new quality values, the hereditary calculation may have the capacity to land at preferable arrangement over was already conceivable. Transformation is an essential piece of the hereditary inquiry as it serves to keep the populace from stagnating at any nearby optima. Change happens amid advancement as indicated by a client perceptible transformation likelihood [34].This probability should usually be set fairly low (0.01 is a good first choice), e.g.

Not mutated chromosome: 1 0 0 0 1 1 1 1

Mutated 1 1 0 1 1 1 1

1.7.7 Advantages of Genetic Algorithm

GA has numerous favourable circumstances over other pursuit methods. These focal points include:

- **Robustness:** GA is computationally basic and effective in the quest for development and is not constrained by prohibitive suppositions of the pursuit space.
- **Intrinsic parallelism:** GA search through populations of points, not single point, which makes them intrinsically parallel.
- **Global:** GA use random operation in their evolution processes that allows a wider exploration of the search space.

These highlights have made GA alluring for utilization inside a more extensive scope of designing trains, and are turned out to be fit for yielding promising results in complex applications.

CHAPTER 2

LITERATURE REVIEW

In this chapter, some of the important techniques related to the research problem are discussed. The review of literature is a very important part as it links up the studies that have already conducted in the same field. It also puts light on the various aspects which have already been accomplished by researchers and gives us a chance to appreciate the evidences that have already been collected by researchers in their studies and supports the researchers in projecting the current research work in proper perspectives. Along with that, the researchers get chance to learn from the experience of the other studies in the same field and can enrich the proposed study. The research methodologies used by different researchers help. A comprehensive review of a few research papers is given below.

Query processing in distributed databases is a complex task due to following two reasons:

- Data must be allocated to different sites.
- It must be efficiently accessed, processed and communicated to meet the desired retrieval and update requirements by user.

Genetic algorithm provides an efficient way to solve the above two problems.

In distributed database systems redundancy of data helps in fault tolerance and recovery but they make distributed processing complex. Query optimization is one of the key fields in distributed database systems. It uses inner joins to reduce the communication cost and improve the performance of system. Lin Zhou, Yan Chen, Taoying Li, Yingying Yu [18] in their paper had analysed the query optimization process based on semi-join operation combined with the practical application. They had also developed a new SDD-1 algorithm which is used for query optimization based on inner join operations.

Query optimization is the key factor in distributed database systems for improving the performance, reliability, efficiency of the system. Xiaofeng Li, Dong Le, Hong Zhi Gao, Lu Yao [14] in their paper had put forward query optimization algorithm on multi relation inner join. Their experiment had proved that algorithm for query optimization on multi relation inner join reduces the data volume of intermediate result and also decreases the overall communication cost.

Query Optimization is an important part of distributed database systems. Fan Yuanyuan, Mi Xifeng [37] in their paper had analyzed a number of optimization algorithms. They had composed another semi-associated database query calculation, which had the information of the moderate results produced from the usage of all sub-query as the unequivocal variable of system cost, and characterizes a capacity to focus the improvement advantages of this calculation. Their exploratory results had demonstrated that the enhanced semi-association inquiry improvement calculation had higher enhancement effectiveness, fundamentally decreases the measure of transitional result information, and viably diminishes the aggregate expense of the system interchanges.

In distributed database systems there are three processes by which data is distributed among various sites, these are: fragmentation, allocation, and replication. Fragmentation process requires empirical knowledge of data access and query frequencies. But Shahidul Islam Khan and Dr. A. S. M. Latiful Hoque [21] had proposed a horizontal fragmentation technique that is capable of taking proper fragmentation decision at the initial stage by using the knowledge gathered during requirement analysis phase without the help of empirical data about query execution. It allocates the fragments properly among the sites of DDBMS.

In query processing in distributed systems the main problem is determining the sequence and the sites for performing the set of operations, if the query is subdivided into sub queries that require operations at geographically distributed databases, such that the operating cost for processing the query is minimized. For that B.M. Monjurul Alom, Frans Henskens and Michael Hannaford [10] had proposed a technique to process the query with minimum inter site data transfer. The proposed system is utilized to figure out which relations are to be apportioned into sections, and where the pieces are to be sent for preparing. The method by and large sections the relations that exist in the predicates (the WHERE condition) of the query. It picks more than one connection to stay divided which abuses parallelism, while recreating alternate relations (barring the divided relations) to the destinations of the divided relations. Thus the communication costs and local processing costs can be reduced due to the reduced size of the fragmented relations and the response time of queries can be improved.

In [5] Rho and March had designed a nested genetic algorithm that iteratively allocates data to nodes and to meet the efficient retrieval and update requirements where to process and access the data. In their nested genetic algorithm there were two genetic algorithms. The outer genetic algorithm addresses the first problem of query processing in distributed databases. That is data allocation to various sites. And the inner genetic algorithm addresses the second problem. That is efficiently accessing and processing the data.

The most important concern in query processing in distributed databases is minimizing the query execution time. So different allocation of sub queries to sites and their execution plans need to be optimized based on query type. This subquery allocation problem is NP-Hard. Therefore, Narasimhaiah Gorla and Suk-Kyu Song [23] had optimized the sub query allocation using genetic algorithm. Their proposed GA procedure was tested with simulation experiments on 20 complex queries. It had been found that GA produced better results in much less time than exhaustive method.

In distributed database design, the most important concern is for allocating data and relational operations (e.g. Select, Project, Join, Union) to various sites. Performance, cost concurrency control etc. must be taken care of while performing retrieval or updating queries at various nodes. In [36] Salvatore T. Walk and Sangkyu Rho, had added to a scientific model which figures out where information will be designated, the level of information replication, which duplicate of the information will be utilized for every recovery action, and where operations, for example, select, venture, join, and union will be performed. It has three stages. In the first place, the arrangement of query is dissected to a situated of document sections (vertical and flat segments) for allotment. Second, every query is deteriorated into a situated of steps, each of which references document sections. This may oblige extra join or union steps if asked for information has been divided. Third, the subsequent parts and inquiries are utilized as info to a numerical model that chooses a base expense information and operation portion. The scientific model considers system correspondence, neighbourhood transforming, and information stockpiling expenses. A hereditary calculation is created to settle this scientific detailing.

Distributed query processing algorithms require data reduction to reduce the communication cost. For reducing the data transfer between sites inner joins are used. Peter Scheuermann, Eugene Inseok Chong [11] in their paper had introduced an efficient join processing algorithm for distributed database systems that makes use of bipartite graphs in order to reduce data communication costs and local processing costs. The bipartite graph represents the tuples joined by two relations. Their algorithm also reduces the relations at each site. They had represented an algorithm that can easily adapt to the changes in system configurations like additional resources available or change in data characteristics.

In distributed databases as data is located at different locations so there is need to join data from different sites to get the desired output. Joins are not always beneficial. Sometimes inner join proves to be more beneficial as it reduces the transmission cost. Manik Sharma, [12] in their paper had analyzed the performance of join and inner join in distributed database system over various parameters like query cost, memory used, CPU cost, input/output cost, Data Transmission, Total Time and Response Time. They had shown that inner joins are beneficial if the transmission cost is of main consideration, otherwise joins are beneficial.

In distributed databases data replication, join node selection, join order, and reduction by inner join all have significant impact on the efficiency of the distributed database system. Rho Sangkyu, T. March Salvatore [4] in their paper had compared the various distributed database design models. They had found that replication was most effective for retrieval intensive and high selectivity situations. Join node selection, join order, and reduction by inner join were most effective for balanced retrieval/update and low selectivity situations. There combination offered only marginal improvement. Their results had also shown that there is trade-off between total operating cost and average response time design criteria.

With the advancement in technology businesses want distributed data processing at any cost. Distributed data processing is a complex task because distributed systems can become very large involving thousands of heterogeneous sites, the state of the distributed systems may change rapidly as load over sites varies with time and new sites are added to the system. Donald Kossmann [35] in his paper had discussed query processing in distributed database and information systems. He had discussed architecture of query processing in distributed database systems including various techniques for joins, intraquery parallelism, reducing communication costs and exploiting caching and replication of data.

As the volume of data is increasing day by day relational databases today are seen with large queries containing many joins. Ordering of joins is very important as improper ordering may have a negative effect on the efficiency of DBMS. Join ordering is NP-Complete. For smaller queries optimal join strategy can be found by dynamic programming. But for larger queries it becomes infeasible. Jim Wilenius [15] in his paper had discussed various approaches like Iterative Improvement, Simulated Annealing, Genetic Algorithms, Two phase optimization etc for producing efficient sub-optimal solutions to the join-ordering problem.

In distributed databases there is communication involved as data is located at different sites. Distributed database systems provide scalability and accessibility due to its architecture. While developing distributed database systems security cannot be compromised as it will cause a risk to integrity of data. Carolyn Mitchell [17] in her paper had discussed various security issues and there solutions for distributed database systems.

Traditional query optimizers assume that complete information about selectivity, resource availability is available at run time. But in case of distributed database systems as data is located at different sites and changes to data are possible at various sites. So static plans produced by traditional optimizers may not be optimal for many of their actual run-time invocations. Richard L. Cole [38] in his paper had proposed an optimization model for creating dynamic plans at compile time using exhaustive search in a dynamic programming framework. But his results had shown that despite using dynamic programming and memorization, dynamic plan optimization is slower than traditional optimization.

In distributed database environment, site task of relations is an imperative undertaking. At the point when there is join operations over different destinations are included then picking the site to convey join operation may have critical effect on the execution. W. Cornell Douglas, S. Yu Philip [20] in their paper had added to an approach to allot relations and focus join destinations all the while. The procedure breaks down inquiries into basic connection polynomial math steps extended with potential message steps and makes connection site and join site task together to enhance execution.

CHAPTER 3
PRESENT WORK

3.1 Problem Definition

Query Optimization in Distributed Databases is gaining popularity due to increasing business demands for distributed environment and due to advancement in technology of networks. Query optimization can be done in a number of ways like exhaustively, randomized, genetically etc. Optimization using genetic algorithm helps in finding the near optimal solution in less amount of time.

Considering the following in distributed database environment:

$R = \{r_1, r_2, \ldots, r_n\}$, a set of fragments,

$S = \{s_1, s_2, \ldots, s_m\}$, network sites,

$Q = \{q_1, q_2, \ldots, q_q\}$, set of sub queries.

As data is fragmented and located at different sites so to get the desired output there is need to join two sub queries located at different sites. So data from one of the site must be transmitted to the site of other. But using full join sometimes incurs extra communication cost when complete relation is not required for join. In such cases to reduce the communication cost involved between two sites inner join is used. But inner join reduction is not always viable approach as sometimes all attributes of relation are required for joins operation. In that case it increases the communication cost.

The present study is a humble effort made in analysing the effect of percentage participation of intermediate fragments of operations evaluating dynamically to minimize various costs like I/O cost, CPU cost and Communication cost of a distributed query. Communication cost is the cost of shipping the query and its results from the database site to the site where the query originated. Also effort is made in allocating full join and inner join operators based on dynamic percentage participation computed at run time which helps in reducing the communication cost involved in executing distributed database query. The main concentration will be on reducing the communication cost involved in transmitting the relations from one site to other in a dynamic environment by using either joins or inner joins or combination of both.

3.2 Research Objectives

The main objectives of this thesis are:

- Simulating dynamic distributed database environment in MATLAB.
- Create a database in MS ACCESS and create a connection string with MATLAB to access the database.
- Using Genetic algorithm for optimizing the results to minimize the communication cost involved in sending data from one site to other.
- Analyzing the effect of using full joins and inner joins as join operator on communication cost.
- Getting the best results with minimum communication cost involved.
- Analyzing communication cost and percentage improvement in reducing the communication cost involved by inner joins for different instances of database that is for dynamic database.

3.3 Methodology adopted for experiment

- ➢ Convert the SQL query to relational algebra query.
- ➢ Represent the relational algebra query into query tree where each node represents different operations like selection, projection, joins etc.
- ➢ Dynamically calculate the percentage participation for nodes containing inner join, left join, right join operator.
- ➢ If (Is percentage participation for inner join (PP_{IJ}) < percentage participation for join (PP_J))

  Then, Set $PP=PP_{IJ}$

  else

  Set $PP=PP_J$
- ➢ Calculate fragment size based on PP for join operation nodes.
- ➢ Give this dynamically computed fragment size for join operations as input to the simulator built in MATLAB.

- Simulated distributed database environment in MATLAB will use Genetic Algorithm to minimize the objective function which is communication cost involved in transferring data from one site to other while performing joins.
- Calculate the percentage reduction in communication cost for inner joins against joins for one instance of the database.
- Dynamically calculate the percentage participation for nodes containing join operator both for joins and inner joins for next instance of database.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 DATA ALLOCATION

Data allocation is process of storing each fragment/ replica at site with optimal distribution, which increases locality of reference. It is one of the major task while designing a distributed database. There are four strategies regarding placement of fragments/ tables, which are as:

i. **Centralized**: This strategy has one database which is stored at a central site and users at various sites access that database. Locality of reference is minimal in this case.

ii. **Partitioned (Fragmented)**: It partitions database into various fragments and each fragment is assigned to one site. All the fragments are disjoint. Locality of reference is high, if fragments are allocated to sites where they are accessed most frequently.

iii. **Complete Replication**: It consists of keeping of copies of complete database at each site. This strategy increases locality of reference, but sometimes problem of inconsistency arises.

iv. **Selective Replication**: This strategy is a combination of above three strategies. Some relations or data items are fragmented to achieve high locality of reference, while some relations which are frequently accessed at more than one site and are not updated frequently, are replicated. This strategy provides benefits of all the above mentioned strategies.

Table 2: COMPARISON OF DIFFERENT ALLOCATION STRATGIES [8]

	Locality of Reference	Reliability and Availability	Performance	Storage costs	Communication costs
Centralized	Lowest	Lowest	Not satisfactory	Lowest	Highest

Partitioned	High	Low for item, High for system	Satisfactory	Lowest	Low
Complete replication	Highest	Highest	Best for read	Highest	High for update, Low for read
Selective replication	High	Low for item, High for system	Satisfactory	Average	Low

Table 2: Comparison of different allocation strategies

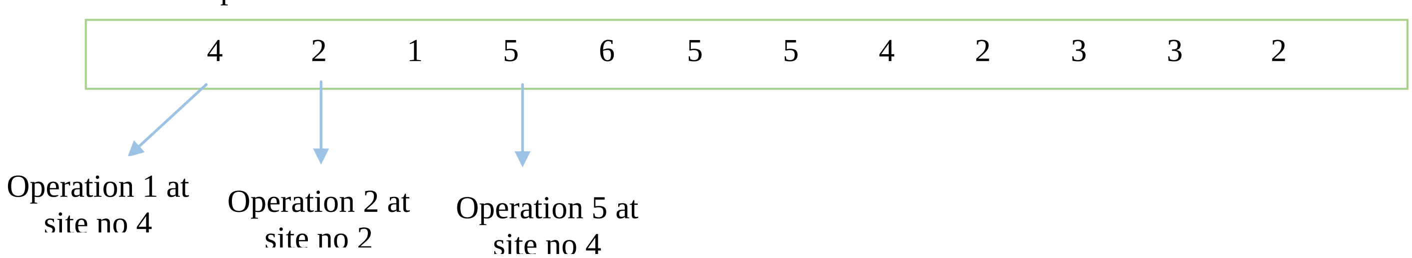
DISTRIBUTED SUB QUERY ALLOCATION

As in case of distributed databases, database relations are allocated on different sites. So, cost incurred in executing a query does not only consists of input-output costs, but communication cost is also there. So, an optimized sub query allocation plan needs be generated which gives such a plan for execution of sub queries that the total cost for query gets reduced.

One such approach based on genetic algorithm [39] devised by Dr. Virk [40] is works as follows:

1. Read the input data file which simulates the distributed environment, by providing allocation plan, communication coefficients etc.

2. Generate an initial population of length equal to number of operations, giving feasible allocation plans. A chromosome is of the form:



3. Calculate the fitness function of each member of the generated initial population. Then rank and sort the population in order of fitness.

4. **Selection Operation**: Select two parents from the population or operation allocation pool without replacement.

5. **Crossover and mutation:** Crossover the selected parent strings and then apply mutation operation to generate a new operation allocation plan. Calculate the fitness of new operation allocation plan.

6. Add the new solutions in the pool and replace the worst from it by replacing it with the best of the previous generation.

7. If the number of generations is less than maximum number of generations, then go to step 4, otherwise print the fittest solution of the final population.

8. Stop.

I have worked on this algorithm. It had been coded in Pascal. I simulated it using java. The environment has been simulated by taking a set *'S'* of data distribution sites, a set *'R'* of relations and a set 'Q' of relations. Let a query *'q'* be broken into a set *'j'* of sub queries on the set of relations *'R'*.

4.2 DECISION VARIABLES USED BY SIMULATOR

a. **Data Allocation Variable:** A_{rs}

A_{rs}=1 (if there is a copy of relation/fragment 'r' at site's')

A_{rs}=1 (if there is a copy of relation/fragment 'r' at site's')

b. **Variables used for site selection for sub query execution:**

S^q_{ys} (sequence of various sites where sub queries gets executed)

S^q_{ys}=1 (if sub query 'y' of query 'q' is done at site's')

S^q_{ys}=0 (otherwise)

c. A notation is proposed for Join operations to handle left previous operation of a join operation (LPO) & right previous operation of a join (RPO) as following:

$S_{yv[p]S}$=1 (for [p]=1 for LPO(left previous operation) of a join)

$S_{yv[p]S}$=1 (for [p]=2 for RPO(left previous operation) of a join)

$S_{yv[p]S}$=0 (otherwise)

d. $\mathbf{I^q_{ry}}$ represents whether the sub query 'y' of query 'q' references intermediate relation/fragment 'r' :

I^q_{ry} =1 (if the base relation 'r' or intermediate fragment 'r' is used by sub query 'y' of 'q' query)

I^q_{ry} =0 (otherwise)

e. Cost Function used:

The cost of processing a query is given by:

$QC_i=LPC_i+CC_i$ (LPC stands for Local Processing Cost, CC: Communication Cost)

4.3 LOCAL PROCESSING COSTS

Local Processing Costs for processing a query's simple selection & projections may be represented as costs of transforming input relation from disk to memory and CPU time for processing a selection or projection at site *S*.

$$LPC_y^q = \sum_s S_{yS}^q \left(IOC_s \sum_r I_{ry}^q M_{ry}^q + CPC_s \sum_r I_{ry}^q M_{ry}^q\right) \qquad (4.1)$$

Where M_{ry}^q *= No. of memory blocks of relations 'r' accessed by sub query 'y' of q.*

IOC_s = Input Output Cost Coefficient of site s in millisecond per 8k bytes

CPC_s= CPU Cost coefficient of site s.

This equation represents input output costs in storing the intermediate results of previous operations to the site of current join operation.

Local processing costs for a join may be given as

$$LPC_y^q = \sum_s S_{yS}^q \left(IOC_s \sum_p \sum_r \rho_p I_{ryv[p]}^q M_{ryv[p]}^q\right.$$

$$+$$

$$\sum_s S_{yS}^q \left(IOC_s \prod_r I_{ry}^q M_{ry}^q + CPC_s \prod_r I_{ry}^q M_{ry}^q\right)$$

Where

‘ρ_p’ ‘Percentage Participation’ & is defined as the ratio of resultant different values of a field to the domain of that field ($0 <= \rho_p <= 1$) .

$M^{q}_{ryv[p]}$ is the size of an intermediate relation.

$v_{[p]}$ represents ‘left previous operation’ of a join for p=1 & ‘right previous operation’ of a join for p=2 .

This equation represents CPU & I/O costs for performing current join operations at site‘s’.

4.4 COMMUNICATION COSTS

These costs are involved in case of join operations and final operation only. As we have assured that selections & projections of retrievals on relations are to be done only at sites which hold a copy of those base relations. Join may be performed at any of all possible sites.

$$\therefore\ COMM^{q}_{y} = \sum_p \sum_s \sum_v S^{q}_{yv[p]S}\, S^{q}_{yv}\, C_{sv} \left(\sum_r I^{q}_{ryv[p]}\, M^{q}_{ryv[p]} \right)$$

Where C_{sv} (is the communication cost coefficient between site s and v)

$C_{sv} = 0$ if $(s = v)$ (i.e. if the previous operations and current join operation is done at the same site)

If the final operation is not done at the query originating/destination site then a Communication Cost component is added separately for costs involved in sending the final query result to the query originating/destination site.

f. Objective Function:

The Objective Function is to: Minimize the sum of all costs incurred: i.e.

$$\min_{s} \left\{ \begin{array}{l} (\sum_s S^{q}_{yS} (IOC_s \sum_r I^{q}_{ry} M^{q}_{ry} + CPC_s \sum_r I^{q}_{ry} M^{q}_{ry})) \quad + \\ (\sum_s S^{q}_{yS} (IOC_s \sum_p \sum_r \rho_p I^{q}_{ryv[p]} M^{q}_{ryv[p]} \quad + \\ \sum_s S^{q}_{yS} (IOC_s \prod_r I^{q}_{ry} M^{q}_{ry} + CPC_s \prod_r I^{q}_{ry} M^{q}_{ry})) \quad + \\ (\sum_p \sum_s \sum_v S^{q}_{yv[p]S} S^{q}_{yv} C_{sv} (\sum_r I^{q}_{ryv[p]} M^{q}_{ryv[p]})) \end{array} \right. \qquad 4.4)$$

4.5 EXPERIMENTAL SETUP

Consider the Department Database for experimental analysis:

Select the details of those employees who are both depositor and student in the department.

SQL Query

Select * from ID,department,student,depositor,salary,employee where

employee.employee_name=depositor.employee_name AND

employee.employee_name=student.employee_name AND

student.salary_number=salary.salary_number AND

salary.department_name=department.department_name AND

department.department_name=ID.department_name

Query Tree

Assuming there are total 10 sites available. Considering that each base relation is allocated to different sites. Query tree for the above query can be drawn as shown in Figure 1.19. From the query tree it can be seen that there are

- selection operations = 7,
- projection operations = 7,
- total join operations = 6.

B1, B2, B3, B4, B5, B6, B7 denotes different base relations allocated to different sites. From the tree it is also clear that one of the base relations (student) is replicated.

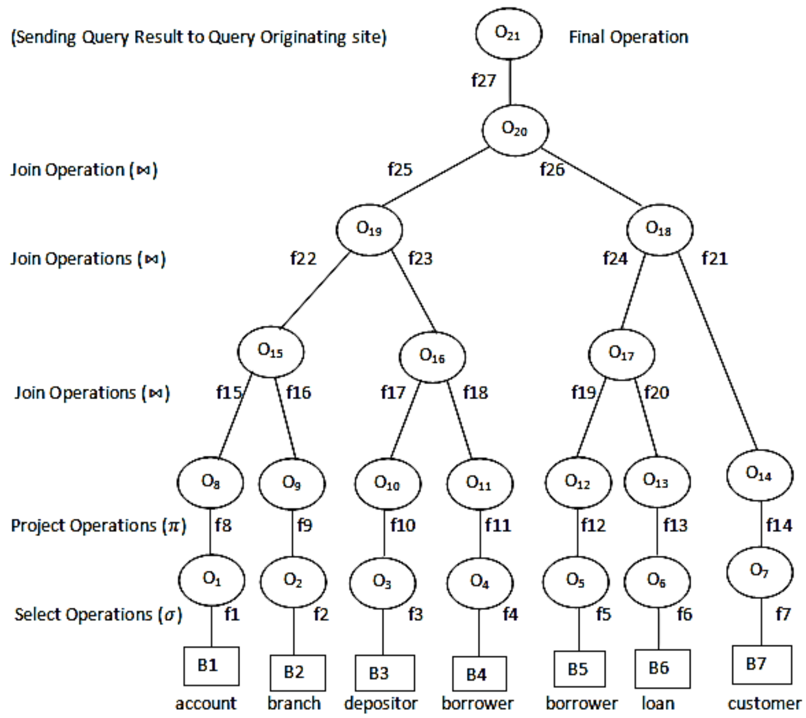


Figure 8: Query Tree for Distributed Database.

Static input provided to the simulator

Communication Coefficients ,I/O Coefficients, CPU Coefficients are assumed to be static in nature and the following table shows the static coefficients provided as input to the simulator.

Table 3: Communication, I/O, CPU Cost Coefficients [29].

Communication Coefficients'	Sites									
	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
S1	0	10	12	13	14	11	12	13	14	11
S2	10	0	11	12	13	14	11	12	13	14
S3	12	11	0	11	12	13	14	11	12	13
S4	13	12	11	0	11	12	13	14	11	12
S5	14	13	12	11	0	11	12	13	14	11
S6	11	14	13	12	11	0	11	12	13	14
S7	12	11	14	13	12	11	0	11	12	13

S8	13	12	11	14	13	12	11	0	11	12
S9	14	13	12	11	14	13	12	11	0	11
S10	11	14	13	12	11	14	13	12	11	0
I/O Coefficients	1	1.1	1.2	1	1.1	1	1.2	1	1.1	1
CPU Coefficients	1.1	1	1	1.1	1	1.2	1	1	1.2	1

Table 4 shows a matrix of 0s and 1s having 1s at those places which represents different operations allocated at different fragments.

Table 4: Intermediate fragments used in various operations [29].

subqueries→ / ↓fragments	SELECTIONS								PROJECTIONS						JOINS						Final Opn.21
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
f1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f7	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
f9	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
f10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
f11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
f12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
f13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
f14	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
f15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
f16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

f17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
f18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
f19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
f20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
f21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
f22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
f23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
f24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
f25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
f26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
f27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Fragment size for selection and projection operations are assumed to remain static during the whole run of simulator, also the size of final operation is kept static and constant. Assume the size of each base relation residing at different sites to be constant as 100 blocks. However the cardinalities of relation could be different. The percentage participation for selection and projection operators are assumed to be constant and are taken as 0.7 for selection and 0.9 for projection and based on these values intermediate fragment size for selection and projection are calculated as follows and these static values are provided to the simulator.

Operation 1: Selection on B1 -- Size: 100 x 0.7 = 70 blocks

Operation 2: Selection on B2 -- Size: 100 x 0.7 = 70 blocks

Operation 3: Selection on B3 -- Size: 100 x 0.7 = 70 blocks

Operation 4: Selection on B4 -- Size: 100 x 0.7 = 70 blocks

Operation 5: Selection on B5 -- Size: 100 x 0.7 = 70 blocks

Operation 6: Selection on B6 -- Size: 100 x 0.7 = 70 blocks

Operation 7: Selection on B7 -- Size: 100 x 0.7 = 70 blocks

Operation 8: Projection on B1 -- Size: 70 x 0.9 = 63 blocks

Operation 9: Projection on B2 -- Size: 70 x 0.9 = 63 blocks

Operation 10: Projection on B3 -- Size: 70 x 0.9 = 63 blocks

Operation 11: Projection on B4 -- Size: 70 x 0.9 = 63 blocks

Operation 12: Projection on B5 -- Size: 70 x 0.9 = 63 blocks

Operation 13: Projection on B6 -- Size: 70 x 0.9 = 63 blocks

Operation 14: Projection on B7 -- Size: 70 x 0.9 = 63 blocks

Operation 21: $f_{27} \rightarrow$ Final Result to Query Site, Size: 10 blocks

4.5.1 Dynamic Percentage participation Calculated by Simulator at run time

Considering the above database schema, SQL query and query tree initially Percentage participation for both joins and inner joins are calculated for Base Relations B1, B2, B3, B4, B5, B6, and B7.

These are calculated by using the following formula [1]

$$PP_J(R, S) = \frac{card(R \bowtie S)}{card(R) * card(S)}$$

$$PP_{IJ} = \frac{card\ (\prod_A (S))}{card\ (dom\ [A])}$$

As inner join is a relational algebra operator so it is implemented in SQL using inner join as it gives the same output as inner join. And simply join is implemented as Full Outer Join as its output is same as simple join in relational algebra.

Decomposing the SQL query and calculating the Percentage participation for various Sub Queries

The above SQL Query can be decomposed into various sub queries. These sub queries using join and inner join are shown below and how simulator calculates their percentage participation are also shown below:

Using Joins

1. Join on $f_{22} = f_{15} \bowtie f_{16}$

   j_sqlquery1='Select * from ID left join department on ID.department_name=department.department_name Union All Select * from ID right join department on ID.department_name=department.department_name';

   $card(ID \bowtie department) = 268$

   $card(ID) * card(department) = 100 * 80$

   $PP_J\,(ID \bowtie department) = \frac{268}{100 * 80} = 0.0335$

2. Join on $f_{23} = f_{17} \bowtie f_{18}$

j_sqlquery2='Select * from depositor left join student on depositor.employee_name=student.employee_name Union All Select * from depositor right join student on depositor.employee_name=student.employee_name';

$card(depositor \bowtie student) = 120$

card(depositor)*card(student) = 70*50

$PP_J(depositor \bowtie student) = \frac{120}{70*50} = 0.0343$

3. Join on $f_{24} = f_{19} \bowtie f_{20}$

j_sqlquery3='Select * from student left join salary on student.salary_number=salary.salary_number Union All Select * from student right join salary on student.salary_number=salary.salary_number';

$card(student \bowtie salary) = 105$

card(student)*card(salary) = 55 *50

$PP_J(student \bowtie salary) = \frac{268}{100*80} = 0.0382$

4. Join on $f_{26} = f_{24} \bowtie f_{21}$

j_sqlquery4='Select * from (student left join salary on salary.salary_number=student.salary_number) right join employee on employee.employee_name=student.employee_name';

$card((student \bowtie salary) \bowtie employee) = 124$

card(student⋈salary)*card(employee) = 123 *105

$PP_J((student \bowtie salary) \bowtie employee) = \frac{124}{123*105} = 0.0096$

Using Inner joins

1. Inner join on $f_{22} = f_{15} \ltimes f_{16}$

sj_sqlquery1='Select * from ID inner join department on ID.department_name=department.department_name';

$card(\prod_{department\_name}(ID \ltimes department)) = 100$

card(dom[department_name]) = 100

$PP_{IJ}(ID \ltimes department) = \frac{100}{100} = 1$

It comes out to be 1 because department_name is foreign key in ID relation and primary key in department relation. If R.A being a foreign key of S (S.A is a primary key). In this case the inner join percentage participations 1 since $\prod_A(S) =$ card(dom[A]) [4].

2. Inner join on $f_{23} = f_{17} \ltimes f_{18}$

   sj_sqlquery2='Select * from depositor inner join student on
   depositor.employee_name=student.employee_name';

   card ($\prod_{employee\_name}$ (depositor $\ltimes$ student)) = 2

   card(dom[employee_name]) = 123

   PP_{IJ} (depositor $\ltimes$student) = $\frac{2}{123}$ = 0.0163

3. Inner join on $f_{24} = f_{19} \ltimes f_{20}$

   sj_ sqlquery3= 'Select * from student inner join salary on
   student.salary_number=salary.salary_number';

   card ($\prod_{salary\_number}$ (student $\ltimes$ salary)) = 50

   card(dom[salary_number]) = 55

   PP_{IJ} (student $\ltimes$salary) = $\frac{50}{55}$ = 0.9091

4. Inner join on $f_{26} = f_{24} \ltimes f_{21}$

   sj_sqlquery4='Select * From (student inner join salary on
   salary.salary_number=student.salary_number) inner join employee on
   student.employee_name=employee.employee_name';

   card ($\prod_{employee\_name}$ ((student $\ltimes$ salary) $\ltimes$ employee)) = 47

   card(dom[employee_name]) = 123

   PP_{IJ} ((student $\ltimes$ salary) $\ltimes$ employee) = $\frac{47}{123}$ = 0.3821

Operations O_{20} and O_{19} are implemented as simple Join not Inner join because as query tree is traversed upwards the selection is getting refined based on conditions therefore the domain of relation is coming down and hence PP_{IJ} will move up that is percentage participation for inner join will keep on increasing as query tree is traversed upwards. But still percentage participation for them are computed as they will be needed to compute the fragment size for join relations further in the code. So O_{20} and O_{19} are implemented simply using joins as follows:

- Join on $f_{25} = f_{22} \bowtie f_{23}$

  sqlquery5='Select * from ID,department,student,depositor where student.employee_name=depositor.employee_name AND depositor.ID_number=ID.ID_number AND ID.department_name=department.department_name';

  $card(f_{22} \bowtie f_{23}) = 2$

  $card(f_{22})*card(f_{23}) = 268*2$

  $PP_J(f_{22} \bowtie f_{23}) = \frac{2}{268*2} = 0.0037$

- Join on $f_{27} = f_{25} \bowtie f_{26}$

  sqlquery6='Select * from ID,department,student,depositor,salary,employee where employee.employee_name=depositor.employee_name AND employee.employee_name=student.employee_name AND student.salary_number=salary.salary_number AND salary.department_name=department.department_name AND department.department_name=ID.department_name';

  $card(f_{25} \bowtie f_{26}) = 2$

  $card(f_{25})*card(f_{26}) = 2*124$

  $PP_J(f_{25} \bowtie f_{26}) = \frac{2}{2*124} = 0.0081$

After calculating the percentage participation, these are provided as input to the simulator which in turn calculates the fragment size for various operations. Simulator is run for three different cases. For all the cases the GA Parameters are kept same.

Case 1: Using Joins as Join Operators

In this case percentage participation for join i.e. PP_J calculated dynamically as shown above are given as input to the simulator and fragment size are calculated for them as shown below:

Operation 15: $(f_{15} \bowtie f_{16}) \rightarrow f_{22}$, Size: 63 x 0.0335 PP_J (1) =2.1105 blocks

Operation 16: $(f_{17} \bowtie f_{18}) \rightarrow f_{23}$, Size: 63 x 0.0342 PP_J (2) = 2.1546 blocks

Operation 17: $(f_{19} \bowtie f_{20}) \rightarrow f_{24}$, Size: 63 x 0.0381 PP_J (3) = 2.4003 blocks

Operation 18: (f_{21}⋈ f_{24}) → $f_{26,}$ Size: 63 x 0.0096 PP_J (4) = 0.6048 blocks

Operation 19: (f_{22}⋈ f_{23}) → $f_{25,}$ Size: 63 x 0.0037 PP_J (5) = 0.2331 blocks

Operation 20: (f_{21}⋈ f_{24}) →$f_{27,}$ Size: 63 x 0.0081 PP_J (6) = 0.5103 blocks

With these fragment size values the simulator is run. Here out of 50 only 2 generations of GA Simulator calculating the costs are shown.

Parameters used:

Size of population: 50 Length of chromosome: 20

Crossover Probability: 0.6 Mutation Probabilities: 0.2

Generation 1

	Chromosomes	Fit Value	I/O Cost	CPU Cost	Comm Cost	Total Cost
1.	8 6 1 8 1 7 7 8 7 10 8 6 10 6 1 2 9 5 9 9	260.591981	143.635728	143.621480	96.484454	383.741662
2.	10 10 1 8 3 10 6 10 1 10 4 10 7 6 7 7 2 2 6 2	260.038307	155.716263	131.614241	97.228222	384.558726
3.	8 3 6 1 9 10 6 2 2 3 9 3 9 6 8 2 10 6 7 4	274.515161	139.855502	132.160568	92.262536	364.278606
4.	8 8 3 7 7 5 3 1 9 2 2 7 9 3 8 4 2 3 7 5	271.736089	140.474023	135.350331	92.179765	368.004119
5.	7 4 4 1 8 4 7 2 9 2 6 5 9 4 8 1 1 7 1 9	271.741094	135.660092	139.861025	92.476224	367.997341
6.	8 1 9 10 10 9 8 5 1 4 4 1 10 6 3 4 4 1 8 10	288.165286	143.351571	139.893890	63.777602	347.023063
7.	2 10 1 8 9 9 1 4 3 9 4 10 2 3 4 2 9 6 6 2	273.644284	143.802433	143.830359	77.805135	365.437927
8.	2 4 1 6 4 2 3 10 7 5 10 2 8 8 6 2 6 9 2 3	258.346482	139.681591	147.416449	99.979034	387.077073
9.	7 7 7 10 1 8 3 2 4 10 4 2 3 4 10 8 10 4 2 7	271.430160	135.977761	131.345890	101.095245	368.418896
10.	10 2 2 2 4 4 3 3 9 5 5 7 8 2 7 5 9 9 3 7	281.654746	152.351056	139.318449	63.375107	355.044612
11.	2 8 5 2 1 7 2 8 3 10 3 5 2 3 1 9 8 2 5 10	265.061571	140.159865	143.038331	94.072638	377.270835
12.	9 5 9 4 8 4 9 8 4 3 8 10 4 7 5 6 7 6 5 7	260.544255	147.862761	139.918449	96.030745	383.811955
13.	7 7 7 10 3 9 3 2 7 5 5 7 8 4 7 5 9 9 7 3	297.532484	152.881056	139.248449	43.968251	336.097755
14.	7 1 1 8 10 6 2 9 4 5 8 1 1 7 7 6 8 8 8 9	264.529710	143.662433	139.719570	94.647367	378.029371
15.	2 4 9 9 1 1 6 6 3 5 10 6 6 9 5 7 7 3 4 10	264.016095	155.964229	132.492786	90.307771	378.764787

16.	1 3 9 1 10 2 5 2 7 7 3 5 1 5 2 2 4 2 5 4	282.790799	144.224865	135.470451	73.922979	353.618295
17.	5 5 4 8 7 8 10 10 2 6 3 4 5 3 7 2 7 1 3 5	261.204259	156.196364	131.345890	95.299896	382.842150
18.	7 4 9 6 4 10 9 6 7 2 7 1 3 9 9 5 4 1 8 6	260.749181	144.041571	144.093129	95.375614	383.510313
19.	6 8 5 7 6 4 8 7 2 2 1 1 5 7 5 6 2 7 5 2	265.562111	143.894557	139.609331	93.055853	376.559741
20.	6 9 6 6 10 7 10 8 4 6 1 1 6 8 10 2 6 3 1 4	281.027402	139.304229	140.260906	76.272051	355.837186
21.	2 4 2 2 2 7 3 7 1 8 5 1 3 10 2 9 6 10 1 5	262.335408	144.232433	147.148786	89.810174	381.191394
22.	7 3 3 10 3 8 3 5 7 5 7 7 8 4 7 5 9 9 3 7	283.551837	152.851056	139.048449	60.769697	352.669202
23.	4 2 6 4 10 7 4 10 5 7 2 4 2 6 2 9 7 4 10 7	256.754427	151.923296	139.683890	97.870032	389.477218
24.	4 1 9 10 7 8 1 4 7 2 8 1 3 1 7 4 6 3 6 9	271.132055	143.750092	143.651480	81.422394	368.823966
25.	9 8 2 1 4 6 4 5 4 5 6 10 7 2 2 5 10 4 6 1	294.673981	143.631571	131.948919	63.777602	339.358092
26.	5 4 8 8 2 2 5 7 8 7 3 7 5 10 2 10 8 6 5 3	273.784343	140.022761	131.140449	94.087771	365.250981
27.	7 5 5 4 8 4 8 7 2 2 1 1 5 7 9 6 2 5 2 2	259.612084	143.98572	147.247331	93.957023	385.190082
28.	10 2 3 4 1 7 5 10 2 7 2 4 2 8 9 6 7 3 6 9	257.038397	148.145092	147.180480	93.721361	389.046933
29.	5 10 1 8 3 3 6 7 5 3 10 10 2 2 2 7 6 1 5 8	257.119551	148.171036	139.273890	101.479212	388.924139
30.	2 1 9 9 9 9 1 6 3 9 8 10 2 3 6 5 10 6 6 2	285.710137	139.907433	140.261359	69.836290	350.005082
31.	9 8 2 6 4 1 4 5 2 3 5 10 3 10 2 10 8 6 1 9	286.478149	139.677433	131.742142	77.647187	349.066763
32.	8 1 9 10 10 9 8 6 2 9 2 1 2 4 3 4 5 7 5 3	263.933620	147.680420	135.920331	95.282394	378.883145
33.	8 5 1 3 5 3 5 1 5 9 1 10 7 10 3 7 3 7 5 4	257.393129	160.463695	131.411451	96.635614	388.510759
34.	2 4 9 2 10 1 6 5 7 7 6 5 1 10 2 2 4 8 7 1	267.631197	143.810502	135.810451	94.027565	373.648517
35.	8 4 3 5 1 2 10 10 6 7 3 8 9 1 1 2 7 5 7 5	270.345832	147.945193	135.550331	86.401060	369.896584
36.	8 6 7 7 1 7 5 7 9 10 8 6 9 6 10 2 9 5 9 3	273.712788	144.076591	139.832241	81.437635	365.346467
37.	6 2 6 2 2 2 8 9 8 4 6 1 2 2 7 5 2 4 8 1	281.630797	151.881571	131.856009	71.337224	355.074804
38.	8 8 3 7 10 4 7 8 5 7 2 10 2 3 8 5 8 5 2 1	284.564489	140.089865	131.131451	80.192875	351.414192

39.	6 4 3 5 5 4 6 8 5 5 2 7 5 9 8 9 6 3 2 7	271.919803	139.900420	147.647331	80.207738	367.755489
40.	5 8 6 5 7 7 7 1 9 4 6 1 3 4 6 6 10 5 3 8	276.878724	135.919331	147.667331	77.582295	361.168957
41.	7 8 2 6 4 6 4 5 2 3 1 10 7 9 2 10 8 6 5 3	286.288272	139.782761	131.850449	77.665066	349.298276
42.	8 5 1 3 2 3 5 6 5 9 6 10 7 10 3 7 3 7 7 1	257.491972	160.468160	131.551451	96.342012	388.361622
43.	6 10 3 7 1 4 7 8 9 10 8 6 10 6 1 2 9 5 9 1	258.572480	143.764865	143.761361	99.212533	386.738759
44.	6 9 6 2 2 5 8 3 8 4 6 1 2 2 7 4 2 5 2 1	289.833490	147.869865	135.880451	61.275374	345.025690
45.	9 5 9 4 8 4 9 8 4 3 10 5 4 7 5 6 5 2 9 3	274.742906	143.846591	139.912241	80.217809	363.976640
46.	9 8 7 9 4 6 3 5 2 3 1 10 7 10 2 10 8 6 5 3	286.097672	140.012761	131.810449	77.707771	349.530981
47.	8 5 8 9 7 5 3 6 2 9 6 5 9 1 8 1 1 3 8 10	267.094170	135.889229	139.799331	98.711225	374.399785
48.	7 8 4 1 8 9 10 1 9 2 2 7 9 6 8 2 10 6 7 9	299.704351	139.716364	131.930687	62.015105	333.662156
49.	7 6 4 1 8 4 7 7 2 2 9 7 5 8 8 10 9 4 7 2	266.771222	139.956364	139.613890	95.282772	374.853026
50.	1 9 10 8 1 3 4 8 2 2 6 5 9 8 4 1 10 1 8 10	274.526719	135.471571	139.753890	89.037809	364.263269

Fitness Sum	Maximum Fitness	Minimum Fitness	Average
1.3612e+004	299.7044	256.7544	272.2441

Generation 50

	Chromosomes	Fit Value	I/O Cost	CPU Cost	Comm Cost	Total Cost
1.	5 6 4 5 7 7 7 3 1 4 6 7 9 9 8 10 1 4 2 3	270.419737	136.117761	135.864890	97.812842	369.795494
2.	8 4 3 1 8 3 4 6 3 7 5 2 8 2 3 3 6 1 2 3	266.458192	152.167761	139.413890	83.711745	375.293397
3.	1 3 8 1 10 8 5 6 3 5 6 6 6 3 10 7 7 4 3 10	265.557349	151.930502	131.795890	92.840103	376.566494
4.	2 8 5 2 4 7 2 6 3 10 3 10 2 9 1 6 7 6 5 7	261.292533	144.107761	143.327449	95.277602	382.712812
5.	5 4 4 8 7 8 10 10 4 2 7 4 9 6 9 5 10 10 3 6	268.040695	143.670502	139.589570	89.817605	373.077677
6.	8 5 3 7 10 5 7 8 5 7 2 10 2 3 8 6 8 4 3 1	281.413027	136.190502	138.974009	80.185066	355.349577
7.	9 1 3 1 5 1 6 2 2 3 10 9 9 1 10 3 1 3 7 5	270.984649	143.929023	136.050331	89.045240	369.024594
8.	8 3 3 3 7 10 10 2 4 4 5 2 10 4 3 10 5 1 10 2	271.591457	148.187433	131.245890	88.766771	368.200094
9.	**6 7 1 7 4 1 5 2 2 3 2 5 7 3 3 1 7 5 6 5**	**277.426837**	**152.481263**	**135.503241**	**72.470890**	**360.455394**

10.	5 4 8 8 2 5 5 7 8 8 3 7 7 2 1 10 10 4 6 9	271.118110	135.922433	135.108039	97.812465	368.842937
11.	2 4 1 6 4 3 1 10 9 2 10 2 8 8 6 2 6 6 2 6	268.298127	139.501036	147.656687	85.561986	372.719709
12.	9 5 9 4 8 4 9 8 4 3 8 10 2 7 5 7 2 2 9 10	274.093446	152.034865	131.904241	80.899972	364.839078
13.	5 7 5 1 7 10 5 4 6 2 10 9 9 3 6 4 5 4 2 2	273.204515	140.041899	143.532890	82.451373	366.026162
14.	3 3 7 3 8 8 3 2 2 1 5 1 4 10 5 9 5 10 4 2	257.385258	148.387433	139.180786	100.954421	388.522641
15.	8 9 10 8 1 3 4 2 3 7 6 7 6 8 6 10 8 2 2 1	267.054736	135.694865	139.649451	99.110755	374.455071
16.	6 6 2 3 4 2 10 7 3 8 2 7 5 3 9 10 9 4 7 2	256.866019	144.191364	147.411890	97.704760	389.308014
17.	4 6 4 1 8 4 7 8 2 2 5 5 2 8 6 2 6 3 2 3	270.769651	139.560420	147.507331	82.249857	369.317608
18.	2 4 1 6 4 2 3 10 7 5 10 2 8 8 8 2 1 5 8 10	271.054052	139.675400	135.500331	93.754374	368.930105
19.	8 6 3 7 1 7 1 8 8 5 5 7 8 1 3 5 9 8 3 7	285.732620	152.162227	139.439331	58.375984	349.977542
20.	5 7 8 10 3 9 3 2 9 10 8 2 10 6 1 10 9 5 9 3	276.915823	140.011591	143.421241	77.687738	361.120570
21.	3 3 7 3 9 10 8 10 6 10 9 9 9 3 6 1 5 4 2 2	270.185596	140.241899	143.702890	86.171167	370.115955
22.	8 4 3 1 8 9 4 6 3 7 5 8 8 2 2 3 6 1 5 2	260.462311	147.901899	139.613890	96.416918	383.932707
23.	4 9 6 2 2 5 8 9 8 4 6 1 2 6 7 5 4 5 4 1	267.417896	147.725400	136.061906	90.159245	373.946551
24.	5 3 10 8 1 3 4 9 2 8 7 9 5 8 8 10 9 4 7 10	278.116238	139.915502	139.453890	80.192498	359.561889
25.	2 9 6 8 1 3 10 7 2 3 2 7 1 8 1 10 6 4 7 2	269.578231	135.861364	143.512890	91.575580	370.949834
26.	1 6 5 1 2 2 5 10 8 9 10 4 2 5 5 2 4 3 4 6	277.276518	143.709229	135.612025	81.329553	360.650807
27.	9 6 8 8 6 10 7 10 1 9 7 10 7 5 9 3 5 10 6 9	258.389873	151.992433	139.782480	95.237158	387.012072
28.	6 9 6 2 2 5 8 4 8 4 6 1 2 2 7 5 2 5 8 1	280.511270	151.820400	131.981451	72.690065	356.491916
29.	5 2 4 8 7 8 10 10 2 2 4 3 6 6 9 5 4 7 5 6	264.323794	143.783695	143.388570	91.151602	378.323867
30.	7 7 9 10 7 6 9 9 7 2 8 1 3 1 1 9 7 4 10 1	256.402182	148.396571	143.823009	97.792702	390.012282
31.	3 4 3 1 8 3 4 4 3 1 1 2 8 2 4 3 6 1 5 2	273.122182	143.966899	143.452890	78.716712	366.136500
32.	8 10 9 9 3 3 6 9 4 6 10 1 3 2 9 7 6 1 10 8	268.656366	148.056571	147.931890	76.234246	372.222707

33.	9 3 5 6 4 6 4 5 2 3 9 10 2 5 5 2 6 3 4 6	271.860864	143.989229	139.911025	83.934963	367.835217
34.	1 5 9 7 2 4 5 10 2 9 7 4 7 10 2 10 8 6 5 3	283.165007	140.012761	131.650449	81.487771	353.150981
35.	7 6 8 1 8 4 7 8 2 9 6 9 9 4 8 7 1 9 7 2	277.116929	143.780193	136.039449	81.038860	360.858502
36.	3 3 7 3 8 10 8 4 3 2 10 5 4 3 6 1 5 4 2 10	280.054644	140.211036	143.082890	73.779245	357.073172
37.	8 4 3 5 1 2 10 8 10 10 4 8 7 6 10 7 2 2 9 2	279.705859	147.730728	131.444241	78.343463	357.518432
38.	3 3 1 9 8 5 8 10 5 9 3 4 4 1 1 2 7 8 7 5	263.013270	148.066364	135.650331	96.492258	380.208953
39.	2 8 4 8 2 7 1 7 7 8 3 1 3 10 2 9 6 4 1 5	263.017386	144.042433	147.183345	88.977224	380.203002
40.	1 2 8 2 2 1 4 7 8 6 8 8 8 10 9 1 4 4 7 6	261.249920	139.435502	147.352129	95.987607	382.775237
41.	9 7 2 9 3 5 4 5 2 3 6 10 7 5 2 10 8 6 5 3	283.027252	140.282761	131.680449	81.359656	353.322866
42.	1 10 8 2 4 4 3 3 9 8 6 2 5 1 7 8 6 4 2 7	274.151551	143.737761	139.623890	81.400100	364.761751
43.	4 2 9 5 5 4 6 8 5 5 2 1 3 4 7 10 8 5 3 8	271.415740	143.849331	131.771331	92.817809	368.438470
44.	6 4 3 3 1 5 3 6 7 4 8 4 7 8 5 1 4 5 3 5	278.024696	139.985193	139.649331	80.045753	359.680278
45.	6 3 1 7 4 1 5 2 2 3 2 5 3 7 3 8 6 10 1 5	268.176564	144.282433	139.470786	89.135442	372.888662
46.	2 8 4 8 2 7 3 7 7 8 5 1 3 10 2 9 9 5 6 5	256.818262	148.276263	147.080241	94.023905	389.380409
47.	1 6 9 7 3 2 5 8 2 9 7 4 2 5 5 2 6 3 2 3	262.269485	144.225420	139.679331	97.382459	381.287210
48.	2 2 1 6 4 2 3 10 7 5 10 2 8 8 6 2 6 3 4 9	258.148333	139.785092	147.309025	100.280067	387.374184
49.	4 10 5 2 8 3 5 2 7 5 6 3 7 8 3 2 3 7 5 6	268.135543	156.448695	131.271570	85.225444	372.945709
50.	7 1 1 1 3 6 2 8 4 3 7 1 1 7 7 3 8 8 8 3	273.259962	152.193296	131.741331	82.017264	365.951891

Fitness Sum	Maximum Fitness	Minimum Fitness	Average
1.3503e+004	285.7326	256.4022	270.0546

Case 2: Using Inner join as Join Operator

In this case percentage participation for inner join i.e. PP_{IJ} calculated dynamically as shown above are given as input to the simulator and fragment size are calculated for them as shown below:

Operation 15: $(f_{15} \ltimes f_{16}) \rightarrow f_{22}$, Size: 63 x 1 PP_{IJ} (1) =63 blocks

Operation 16: ($f_{17} \ltimes f_{18}$) → f_{23}, Size: 63 x 0.0162 PP_{IJ} (2) = 1.0206 blocks

Operation 17: ($f_{19} \ltimes f_{20}$) → f_{24}, Size: 63 x 0.9090 PP_{IJ} (3) = 57.267 blocks

Operation 18: ($f_{21} \ltimes f_{24}$) → f_{26}, Size: 63 x 0.3821 PP_{IJ} (4) = 24.0723 blocks

Operation 19: ($f_{22} \ltimes f_{23}$) → f_{25}, Size: 63 x 0.0037 PP_{J} (5) = 0.2331 blocks

Operation 20: ($f_{21} \ltimes f_{24}$) → f_{27}, Size: 63 x 0.0081 PP_{J} (6) = 0.5103 blocks

With these fragment size values the simulator is run. Here again out of 50 only 2 generations of GA Simulator calculating the costs are shown.

Generation 1

	Chromosomes	Fit Value	I/O Cost	CPU Cost	Comm Cost	Total Cost
1.	6 5 7 7 3 6 2 6 3 3 7 1 7 10 1 4 10 9 9 9	230.247574	153.594561	156.838706	123.881840	434.315108
2.	1 8 4 1 6 6 7 1 3 10 7 1 8 5 7 5 3 2 4 2	224.613427	171.799479	147.622658	125.787223	445.209360
3.	2 10 1 8 2 1 2 10 8 6 9 7 9 5 9 4 6 7 8 6	225.337629	155.647178	165.869160	122.262183	443.778521
4.	4 6 5 8 5 7 9 5 8 6 7 5 10 5 3 2 5 5 1 6	245.153411	168.003617	147.422897	92.481333	407.907847
5.	3 7 8 1 1 10 1 1 9 7 5 3 8 7 3 10 7 3 4 8	225.001721	168.242178	147.292658	128.906210	444.441046
6.	5 8 6 10 5 6 8 2 1 2 6 5 6 1 1 6 8 6 10 10	246.171313	151.055056	158.259995	96.906124	406.221174
7.	8 3 6 8 8 1 9 1 6 4 3 6 3 9 9 10 3 10 8 6	230.287388	163.580056	154.402160	116.257803	434.240019
8.	3 6 1 9 4 10 6 1 8 3 5 2 5 4 6 2 8 8 5 6	232.238733	156.850138	154.282160	119.459097	430.591396
9.	3 2 10 7 8 8 10 8 2 10 1 4 4 4 7 10 2 3 1 9	240.538933	163.758040	147.062897	104.912178	415.733115
10.	2 10 9 7 1 1 10 1 4 7 2 10 7 5 7 10 3 10 1 4	222.684597	167.845056	147.322777	133.897803	449.065636
11.	10 10 9 8 1 8 7 5 7 4 1 6 7 10 8 9 10 6 5 6	223.856332	156.850138	154.051234	135.813712	446.715084
12.	4 5 1 3 8 7 6 7 8 7 2 10 7 5 7 10 2 4 3 4	233.094099	166.910220	145.868578	116.232493	429.011290
13.	2 5 7 7 6 7 7 6 2 10 10 6 6 8 8 10 4 2 2 10	249.622741	153.303699	149.992922	97.307906	400.604526
14.	9 1 1 2 10 6 10 4 1 9 2 7 3 9 5 8 10 7 8 8	236.959015	155.687178	146.423922	119.902810	422.013909
15.	5 7 9 6 6 6 9 5 8 6 3 1 9 7 7 2 3 10 2 10	228.950031	173.500138	146.753922	116.522469	436.776529
16.	9 2 6 8 10 8 5 2 6 5 6 3 2 7 1 8 4 8 10 8	234.032553	151.435056	154.021922	121.834007	427.290985

17.	9 1 7 8 10 8 5 2 6 5 6 3 9 9 7 2 3 4 6 10	231.157710	171.940056	149.085930	111.579096	432.605082
18.	3 6 1 9 6 10 6 7 8 3 3 2 4 4 6 2 6 10 8 5	225.349493	155.530918	162.319922	125.904318	443.755159
19.	9 1 7 10 10 8 5 1 6 5 6 5 9 7 7 8 3 2 6 10	226.920973	167.903617	148.951394	123.827041	440.682052
20.	4 1 2 6 1 4 1 10 6 4 3 7 7 10 7 2 10 10 7 8	241.624767	166.370220	146.313922	101.180716	413.864858
21.	10 9 8 8 1 3 7 10 2 6 3 4 10 8 9 10 9 9 5 3	225.896246	161.135424	161.918995	119.626688	442.681106
22.	6 7 1 4 9 8 4 5 5 4 1 3 1 2 7 5 6 9 9 9	221.197565	165.169561	157.138706	129.776275	452.084542
23.	1 4 4 1 6 6 3 1 3 10 9 6 8 10 7 5 3 6 4 2	221.638749	171.530918	148.061731	131.591994	451.184643
24.	4 5 6 9 8 1 5 9 8 2 4 8 3 4 1 10 9 10 1 9	230.089071	155.320918	159.569897	119.723483	434.614298
25.	9 5 9 10 5 3 9 5 1 6 6 5 6 5 5 6 5 9 10 3	260.090177	159.875342	154.560995	70.045697	384.482033
26.	10 3 5 5 2 6 9 8 1 5 10 7 6 5 3 5 8 6 7 10	237.605311	166.730220	146.142995	107.992799	420.866014
27.	2 7 2 8 6 1 4 5 4 5 4 10 8 7 9 7 10 4 6 3	230.287512	163.611781	156.703930	113.924075	434.239786
28.	6 1 10 10 10 8 5 2 6 5 4 3 4 7 7 2 3 4 9 8	232.042290	173.100138	148.805930	109.049858	430.955927
29.	7 6 1 2 1 4 3 6 7 4 3 3 1 2 1 1 9 9 5 3	225.165906	157.320424	162.188995	124.607552	444.116971
30.	10 10 3 10 7 1 8 2 8 7 3 6 9 5 7 7 4 7 8 6	234.506379	168.172178	149.753160	108.502294	426.427632
31.	2 10 6 3 2 1 2 8 8 1 9 5 9 6 9 9 8 7 8 6	232.712975	159.802178	162.070160	107.841562	429.713900
32.	4 6 3 2 7 10 9 2 6 7 3 3 3 5 7 7 8 7 6 6	236.313176	168.582178	148.801633	105.783453	423.167263
33.	9 9 10 9 3 3 6 6 6 2 3 2 10 7 1 2 1 5 7 6	239.824213	158.908781	154.422160	103.641134	416.972075
34.	5 9 9 3 1 6 7 2 5 3 7 5 1 1 2 4 5 9 5 8	241.580614	161.613699	150.341995	101.984806	413.940500
35.	2 4 1 9 6 10 6 2 8 10 4 5 7 9 2 8 1 4 8 9	242.865101	155.290918	150.447697	106.012597	411.751213
36.	10 9 2 2 10 4 9 7 10 3 3 8 5 4 6 2 3 10 8 6	226.375600	163.820056	153.912160	124.011506	441.743722
37.	9 10 9 2 1 6 7 1 5 3 9 4 1 1 2 10 5 9 5 5	229.288675	161.204561	146.652995	128.273884	436.131441
38.	4 7 6 9 8 3 5 3 10 2 4 7 3 8 1 5 9 10 1 9	229.858809	159.925918	159.259897	115.863860	435.049675
39.	9 1 7 8 10 8 5 2 6 8 6 3 9 7 7 2 3 5 6 5	231.258418	172.069479	148.881394	111.465819	432.416692

40.	5 5 6 7 7 7 9 3 2 10 10 7 7 9 3 8 10 7 6 8	236.565312	160.552178	148.701394	113.462671	422.716242
41.	5 1 6 2 7 9 6 8 7 9 5 4 9 9 1 4 4 4 5 6	242.540274	153.025138	158.565697	100.711825	412.302660
42.	3 8 2 10 5 3 3 8 1 2 6 2 6 5 1 6 8 6 10 1	244.491311	151.655056	157.790114	99.567320	409.012490
43.	8 6 7 8 5 1 5 6 1 10 2 2 2 4 4 9 3 7 10 5	223.514748	163.798040	157.890922	125.708809	447.397771
44.	8 10 10 2 1 4 6 8 3 2 9 7 4 8 3 4 9 1 3 10	236.582645	166.370220	157.985458	98.329596	422.685274
45.	4 2 1 3 3 3 10 1 7 4 2 6 7 10 8 8 5 6 5 6	231.180255	157.250138	146.013234	129.299522	432.562893
46.	2 6 7 9 1 1 4 8 2 2 9 7 4 8 3 4 9 1 2 10	224.521541	165.140138	158.285458	121.965966	445.391562
47.	3 6 8 5 10 1 9 3 4 8 1 5 4 6 1 9 2 9 7 4	221.372048	162.493781	158.290114	130.944318	451.728214
48.	3 8 2 6 2 5 2 9 6 1 10 6 8 2 5 2 4 8 5 8	230.940171	161.035138	150.062922	121.914524	433.012584
49.	6 8 4 6 2 5 2 9 6 9 10 6 8 9 5 2 4 8 9 8	221.951682	160.805138	153.330394	136.412981	450.548512
50.	6 5 5 7 3 5 10 9 3 2 7 3 7 10 10 4 10 9 9 5	230.932157	153.634561	152.599467	126.793582	433.027610

Fitness Sum	Maximum Fitness	Minimum Fitness	Average
1.1627e+004	260.0902	221.1976	232.5406

Generation 50

	Chromosomes	Fit Value	I/O Cost	CPU Cost	Comm Cost	Total Cost
1.	3 6 8 7 4 10 8 6 2 9 4 1 5 4 5 10 8 1 6 3	230.620434	155.351781	149.015930	129.245209	433.612920
2.	1 1 6 4 9 5 10 5 6 8 6 5 1 7 3 9 6 8 6 1	220.219104	163.410056	165.087513	125.595642	454.093211
3.	2 7 5 8 6 1 2 8 4 5 5 10 8 7 9 8 5 2 9 8	230.968399	161.203699	156.469394	115.286570	432.959663
4.	1 10 10 4 10 4 1 3 5 5 2 6 1 10 1 1 8 6 6 2	228.734056	150.995918	156.838467	129.354557	437.188943
5.	4 1 2 5 10 4 1 7 6 4 2 7 7 6 7 5 10 10 5 7	245.642506	165.081863	146.153922	95.859884	407.095669
6.	5 7 8 7 9 1 7 2 6 7 1 2 6 7 2 1 10 8 6 8	242.706142	155.960056	152.780394	103.280438	412.020887
7.	6 3 9 5 9 10 10 10 9 1 1 3 4 5 2 5 7 4 8 2	220.500245	167.805918	146.418458	139.289858	453.514235
8.	3 8 4 3 2 5 2 9 6 9 10 6 8 9 9 3 4 8 5 8	226.551723	165.300138	158.110922	117.989247	441.400307
9.	8 9 3 10 7 9 5 6 8 2 5 5 2 10 5 2 6 10 4 9	237.464813	159.815918	155.260897	106.038207	421.115022

10.	8 6 8 1 6 1 2 1 5 1 8 7 6 4 10 3 2 4 5 6	234.761631	164.700138	146.418697	114.845151	425.963986
11.	8 6 8 5 10 2 9 2 3 8 1 1 1 3 1 9 5 9 7 4	221.470103	162.533781	158.050114	130.944318	451.528214
12.	4 6 7 5 9 6 3 4 6 10 5 2 4 10 8 5 2 5 4 6	246.810627	159.703617	147.762897	97.702427	405.168940
13.	3 2 4 6 2 5 2 9 6 9 10 6 8 9 4 2 4 8 5 8	225.818054	157.010138	154.341922	131.482329	442.834389
14.	8 3 6 8 9 1 9 1 6 4 3 6 10 9 9 10 8 8 5 9	229.968511	156.811000	154.602160	123.428982	434.842143
15.	2 5 6 7 7 7 9 6 2 10 10 6 7 9 4 8 7 7 8 8	247.550367	160.272178	150.192922	93.493098	403.958197
16.	9 5 9 1 1 4 7 1 9 3 9 4 1 1 10 2 4 3 5 8	239.961356	157.237260	150.632922	108.863587	416.733768
17.	6 7 1 4 9 4 4 5 5 4 3 3 1 7 1 1 7 5 1 6	232.214892	159.823617	155.660897	115.151090	430.635603
18.	4 6 9 1 5 7 9 5 8 6 7 5 10 1 3 2 9 9 1 3	225.783523	167.935342	155.859731	119.107042	442.902115
19.	2 6 5 8 5 7 9 5 8 6 7 2 10 8 7 2 3 8 5 10	228.437820	173.460138	145.943922	118.351825	437.755884
20.	7 8 8 2 2 9 2 2 3 7 2 5 10 5 4 2 7 5 1 6	240.298990	164.218617	150.951897	100.977719	416.148233
21.	9 3 7 7 4 5 10 3 1 4 8 3 9 3 6 7 1 10 2 4	222.872993	161.585138	157.891041	129.209858	448.686037
22.	1 6 9 6 7 4 10 8 7 7 5 5 6 1 8 3 3 7 6 5	231.197880	168.003040	149.171394	115.355483	432.529918
23.	6 7 1 4 9 4 4 5 5 2 4 8 3 7 1 7 9 10 1 1	225.739270	163.720056	159.559777	119.709107	442.988940
24.	2 7 6 9 5 3 5 3 10 4 3 3 1 2 1 1 9 9 5 3	225.135490	157.620424	161.948995	124.607552	444.176971
25.	1 6 1 10 6 10 6 1 8 3 3 2 5 4 6 4 3 10 8 5	226.954046	159.255918	158.250922	123.110993	440.617833
26.	3 7 9 2 1 2 6 8 6 4 2 7 9 3 7 6 7 6 6 10	235.045498	168.145056	157.078467	100.226021	425.449544
27.	3 6 4 6 8 10 2 9 6 1 5 6 8 5 5 2 4 8 5 9	231.181514	160.806000	150.363160	121.391376	432.560537
28.	8 3 6 8 9 1 9 1 3 4 3 6 3 9 8 8 5 8 5 7	231.958462	157.091863	146.523922	127.495886	431.111671
29.	10 3 8 3 8 2 10 9 5 6 9 6 4 2 2 6 1 8 1 7	233.879683	155.521781	159.319658	112.728835	427.570274
30.	**9 1 7 8 10 8 5 2 6 5 6 3 9 7 7 6 3 10 6 10**	**245.557304**	**167.915056**	**156.819394**	**82.502469**	**407.236919**
31.	7 2 8 2 1 9 6 9 9 8 10 6 8 9 8 2 4 8 5 8	228.345793	156.910138	150.432922	130.589247	437.932307
32.	3 8 4 6 2 5 2 2 3 9 10 2 2 8 7 2 3 3 9 1	225.593498	173.757260	148.601513	120.916413	443.275186

33.	2 4 2 3 3 5 6 10 7 5 8 3 9 7 7 1 7 10 6 10	227.911933	168.285056	152.570394	117.910518	438.765968
34.	5 1 7 8 10 8 5 2 7 5 6 9 3 9 5 1 10 2 4 8	228.599053	155.828617	151.271658	130.346857	437.447131
35.	7 8 8 2 6 9 6 2 2 9 2 9 6 9 5 6 10 2 5 8	229.122555	157.148699	154.501922	124.797027	436.447648
36.	2 3 2 3 10 6 6 10 4 5 8 4 3 8 7 2 3 3 4 1	235.132841	172.187178	147.322777	105.781550	425.291505
37.	2 5 8 8 7 4 3 10 3 8 5 3 3 10 4 1 2 3 5 6	228.481682	157.547260	153.442160	126.682427	437.671847
38.	4 6 7 5 9 6 3 5 6 10 5 2 8 10 4 6 2 4 4 6	240.304102	155.550056	159.594433	100.994892	416.139381
39.	4 6 5 8 5 3 5 3 10 2 4 3 10 7 1 10 9 10 1 5	232.496337	155.730918	159.059658	115.323727	430.114303
40.	4 5 6 9 1 7 9 5 3 6 7 3 10 5 3 1 7 5 1 6	240.549308	168.213617	151.691897	95.809671	415.715184
41.	2 3 2 2 3 5 6 10 4 5 8 9 7 9 3 8 10 7 8 8	245.042790	160.242178	145.953922	101.895892	408.091991
42.	2 5 6 7 7 7 9 6 2 10 10 6 3 9 5 1 4 2 4 8	230.226087	156.048617	155.540658	122.766369	434.355643
43.	10 3 8 3 5 2 10 9 8 10 3 2 5 2 6 2 5 10 8 5	233.360294	159.855918	153.481922	115.184075	428.521915
44.	3 6 1 9 6 10 6 1 5 6 10 8 4 2 4 6 1 8 1 7	231.827801	151.086781	163.838658	116.429212	431.354651
45.	5 9 7 9 1 9 5 2 7 6 3 4 2 8 9 5 6 9 5 9	225.191673	161.544561	162.319234	120.202358	444.066153
46.	8 8 8 2 1 9 6 2 3 9 4 2 6 8 7 6 3 3 9 1	223.634506	169.292260	156.949513	120.916413	447.158186
47.	4 1 6 9 9 5 10 5 8 8 6 5 10 7 3 9 4 8 3 2	227.841151	166.431083	158.250922	114.220273	438.902277
48.	3 8 4 7 2 5 2 9 5 9 10 6 10 9 5 2 4 7 5 8	237.913287	161.532260	150.032922	108.756026	420.321207
49.	9 1 7 8 10 8 5 2 6 5 7 3 9 7 3 2 3 10 7 6	230.465339	175.070220	145.984160	112.850346	433.904726
50.	2 3 8 9 5 1 3 10 3 8 5 3 1 1 4 1 2 4 9 10	227.319582	157.250138	156.603930	126.055239	439.909308

Fitness Sum	Maximum Fitness	Minimum Fitness	Average
1.1595e+004	247.5504	220.2191	231.9073

Case 3: Using Combination of Joins and Inner joins

In this case as percentage participation for joins as well as inner joins are computed the minimum out of them are provided to the simulator. That is

begin

for i=1:4

If $PP_{IJ}(i) < PP_J(i)$

$PP(i) = PP_{IJ}(i)$

Else

$PP(i) = PP_J(i)$

End

Therefore PP(1) = 0.0335, PP(2) = 0.0343, PP(3) = 0.0382, PP(4) = 0.0096, PP(5) = 0.0037, PP(6) = 0.0081

Percentage participation for joins are computed above. Fragment size for intermediate fragments is calculated as follows:

Operation 15: $(f_{15} \bowtie f_{16}) \rightarrow f_{22,}$ Size: 63 x 0.0335 PP(1) = 2.1105 blocks

Operation 16: $(f_{17} \ltimes f_{18}) \rightarrow f_{23,}$ Size: 63 x 0.0162 PP(2) = 1.0206 blocks

Operation 17: $(f_{19} \bowtie 20) \rightarrow f_{24,}$ Size: 63 x 0.0381 PP(3) = 2.4003 blocks

Operation 18: $(f_{21} \bowtie f_{24}) \rightarrow f_{26,}$ Size: 63 x 0.0096 PP(4) = 0.6048 blocks

Operation 19: $(f_{22} \bowtie f_{23}) \rightarrow f_{25,}$ Size: 63 x 0.0037 PP(5) = 0.2331 blocks

Operation 20: $(f_{21} \bowtie f_{24}) \rightarrow f_{27,}$ Size: 63 x 0.0081 PP(6) = 0.5103 blocks

With these fragment size values the simulator is run. Here also out of 50 only 2 generations of GA Simulator calculating the costs are shown.

Generation 1

	Chromosomes	Fit Value	I/O Cost	CPU Cost	Comm Cost	Total Cost
1.	1 3 1 8 10 3 1 10 4 1 10 8 1 1 9 3 2 9 3 3	263.561056	151.652200	139.529688	88.236837	379.418726
2.	6 1 2 10 8 9 5 8 2 7 8 5 1 5 9 5 6 5 5 1	261.987297	143.706010	147.453484	90.538403	381.697896
3.	3 10 3 10 7 3 7 10 8 6 3 3 2 2 3 2 8 6 6 2	268.545275	148.562110	131.154598	92.659980	372.376688
4.	4 10 5 3 2 1 2 2 10 2 10 6 7 3 10 8 3 6 8 10	278.802656	143.966248	131.351688	83.358704	358.676640
5.	9 3 1 4 9 1 2 8 7 10 3 2 6 3 1 2 5 3 3 1	267.059530	144.055772	135.816484	94.576093	374.448349
6.	8 6 10 3 5 3 9 1 8 6 10 7 7 1 3 5 3 2 2 4	262.097895	156.221010	131.687484	93.628337	381.536830
7.	8 10 9 8 10 2 4 7 4 2 3 3 2 1 3 10 3 9 8 5	274.243992	152.002407	131.451688	81.184704	364.638799
8.	4 10 3 10 7 3 9 1 7 6 5 4 1 7 5 8 7 1 4 8	264.955497	148.191248	131.660981	97.569644	377.421873
9.	4 7 4 3 7 8 3 10 4 3 5 8 6 7 6 1 2 2 4 4	261.165422	140.106545	143.325939	99.466598	382.899081

10.	9 1 8 5 6 4 7 5 1 7 1 6 1 4 3 2 3 1 10 1	264.598637	155.941248	132.029646	89.960003	377.930896
11.	6 9 1 7 1 8 6 9 10 2 10 1 10 3 7 9 7 10 9 4	256.053695	155.915713	139.958394	94.668979	390.543085
12.	1 7 3 9 7 1 8 4 7 2 8 9 1 3 8 6 5 4 1 7	262.516063	140.07297	139.628981	101.227117	380.929071
13.	7 6 10 3 10 1 3 9 4 7 10 4 3 9 5 6 3 9 5 3	273.465174	148.172735	139.669688	77.834854	365.677277
14.	3 1 4 6 8 6 1 10 8 9 1 10 7 6 1 7 3 3 5 8	260.652303	151.661307	136.026364	95.965184	383.652855
15.	4 8 10 8 2 8 2 5 5 4 3 10 2 4 7 6 7 7 7 8	260.447158	151.805772	139.185364	92.963908	383.955045
16.	8 2 4 5 5 5 4 9 9 9 3 10 7 1 5 3 9 3 4 2	268.219285	152.137704	139.636819	81.054746	372.829269
17.	1 8 2 8 6 9 4 10 6 4 8 5 7 2 10 5 10 8 5 9	285.105371	139.436576	131.817603	79.493335	350.747513
18.	1 9 9 2 2 5 2 10 9 5 6 3 3 9 3 9 10 8 4 3	257.726428	148.232973	139.866819	99.908534	388.008326
19.	4 7 4 3 4 8 3 10 4 3 3 8 9 7 6 5 2 4 4 1	263.254002	144.136248	139.459101	96.265923	379.861272
20.	6 9 10 9 9 1 4 8 10 4 3 5 7 2 7 9 2 1 2 5	261.038117	151.961576	140.017526	91.106714	383.085815
21.	7 6 10 3 10 7 10 1 4 6 1 3 3 8 7 6 3 6 7 6	295.627560	152.125178	139.499927	46.638350	338.263455
22.	1 4 10 9 9 1 6 3 4 7 5 4 7 6 8 7 3 3 5 8	274.964168	151.941307	132.187364	79.555080	363.683751
23.	9 8 8 4 9 8 4 10 3 9 6 6 1 10 4 8 10 7 9 1	275.797170	135.281307	136.069394	91.234600	362.585301
24.	6 7 10 1 7 5 1 4 10 1 10 7 5 3 6 10 9 7 4 1	271.417026	139.811842	147.424939	81.199944	368.436724
25.	9 1 2 5 6 4 7 9 6 3 9 4 7 6 4 3 3 9 5 8	273.427427	152.166010	136.210688	77.351061	365.727759
26.	7 1 10 3 2 1 3 3 4 6 5 6 1 6 4 2 10 2 4 2	280.581576	139.867407	135.737819	80.797363	356.402589
27.	5 5 9 8 10 2 1 7 4 2 10 4 4 7 6 8 7 7 4 8	265.733996	143.806842	139.456819	93.052511	376.316171
28.	8 10 5 10 10 3 8 4 4 10 8 1 3 3 9 1 9 7 1 9	258.427625	143.637704	151.064058	92.253774	386.955536
29.	6 9 10 9 9 1 4 6 3 4 3 9 7 8 9 2 1 5 8 10	260.997928	143.841545	144.264364	95.038896	383.144804
30.	5 9 8 5 8 6 8 8 8 8 6 9 1 1 7 9 2 1 2 4	263.414870	151.610713	139.767646	88.250930	379.629289
31.	2 9 10 9 9 1 4 8 10 4 3 5 1 4 3 5 2 1 10 3	267.527466	151.847973	132.019526	89.925897	373.793396
32.	6 7 10 10 10 3 4 7 4 7 8 4 7 2 9 9 8 1 2 4	261.434470	143.940713	147.325646	91.238673	382.505031

33.	5 9 2 8 6 9 4 10 8 4 8 1 10 2 10 7 10 3 8 9	297.359986	143.527704	131.847603	60.917417	336.292725
34.	7 2 7 10 1 2 6 7 4 10 7 7 8 5 2 4 4 7 2 8	266.470024	140.056307	139.315364	95.905061	375.276732
35.	1 8 5 2 8 5 4 6 1 5 6 3 3 9 10 9 10 8 4 3	263.679405	139.672973	139.636819	99.938637	379.248429
36.	6 1 2 10 2 9 1 10 4 1 3 9 1 8 9 3 2 6 8 3	273.818736	151.747973	139.899688	73.557443	365.205104
37.	8 6 6 1 2 7 7 7 4 8 7 3 8 8 8 3 10 8 4 6	269.221322	143.966248	131.579058	95.896300	371.441606
38.	6 9 8 8 2 10 3 7 4 2 10 6 7 10 1 10 9 8 8 4	273.484483	139.701248	143.524484	82.425726	365.651458
39.	1 7 3 5 2 1 1 4 7 5 7 7 3 3 6 8 3 9 8 6	276.250690	144.421545	139.319927	78.248575	361.990047
40.	9 7 3 7 7 9 6 9 4 2 8 9 2 8 8 10 5 4 9 7	268.672156	140.237438	131.962436	100.000957	372.200832
41.	9 8 8 4 9 8 6 6 1 7 5 4 7 6 6 7 3 3 5 8	282.708253	151.801307	140.065364	61.854874	353.721545
42.	10 6 10 3 10 1 3 3 4 9 6 6 5 3 6 10 9 7 6 2	259.063787	139.782704	147.676274	98.546342	386.005321
43.	4 10 3 9 7 1 6 4 7 5 8 3 1 3 5 7 5 4 1 7	265.642154	152.227973	131.750981	92.467324	376.446278
44.	1 7 8 5 7 3 9 7 8 6 3 3 2 2 8 2 3 6 8 10	293.545243	148.501248	131.451688	60.710053	340.662989
45.	8 10 3 8 3 2 4 7 4 2 5 4 8 7 5 8 7 7 4 8	269.481441	148.071842	131.248819	91.762407	371.083068
46.	1 5 7 9 6 7 3 4 4 6 1 10 7 6 2 5 4 1 1 8	266.694469	143.986248	135.969981	95.004678	374.960907
47.	6 2 10 6 10 3 4 7 4 1 8 1 1 4 5 2 2 2 3 4	275.729575	147.595475	131.857484	83.221231	362.674190
48.	9 1 8 5 6 4 7 5 1 7 8 6 1 4 7 5 6 2 10 3	273.922961	147.758270	139.895364	77.412513	365.066147
49.	1 9 2 8 6 6 3 2 10 8 1 9 9 1 4 10 10 7 1 5	277.531868	135.477704	136.097819	88.743458	360.318981
50.	3 2 6 5 1 9 4 3 8 4 7 4 10 2 10 5 10 8 8 9	286.615153	139.802110	131.747603	77.350193	348.899906

Fitness Sum	Maximum Fitness	Minimum Fitness	Average
1.3505e+004	297.3600	256.0537	270.0948

Generation 50

	Chromosomes	Fit Value	I/O Cost	CPU Cost	Comm Cost	Total Cost
1.	1 3 1 8 4 7 5 9 9 10 7 5 2 1 4 8 7 3 3 1	272.451999	143.985772	135.626484	87.424875	367.037130
2.	3 10 2 1 2 5 2 4 3 1 8 2 1 5 5 1 6 5 5 1	277.005320	139.841010	143.224484	77.938403	361.003896

3.	1 9 2 8 6 10 8 10 4 4 7 3 10 5 8 3 2 8 4 10	265.319619	147.691248	131.648819	97.563836	376.903903
4.	6 8 8 1 1 10 10 7 4 2 8 6 10 2 10 3 10 1 8 9	282.945375	143.327110	131.619765	78.478234	353.425109
5.	9 9 4 2 3 1 9 8 10 3 10 3 10 7 9 1 3 2 10 10	272.913178	148.166545	143.714364	74.535989	366.416897
6.	7 3 5 2 9 8 1 1 4 1 7 6 6 2 10 4 9 1 7 9	263.230450	139.871041	143.706765	96.317453	379.895259
7.	9 9 9 2 5 10 6 9 7 7 9 6 1 4 4 2 10 2 4 4	270.665316	139.876545	136.337939	93.245488	369.459971
8.	9 1 8 5 6 5 7 5 1 4 7 3 1 7 1 8 1 2 6 10	292.883539	135.851545	139.658274	65.922822	341.432641
9.	9 3 1 4 8 3 2 9 7 2 4 10 10 1 3 5 6 2 2 4	257.945444	148.031010	139.625484	100.022384	387.678877
10.	2 6 10 3 5 6 4 1 5 7 10 7 7 3 3 2 5 3 3 1	265.380027	152.285772	131.577484	92.954854	376.818109
11.	5 4 5 3 2 1 2 2 10 2 1 6 7 3 5 8 5 4 10 7	271.828231	144.067973	131.419526	92.391877	367.879376
12.	1 7 3 5 7 2 1 9 7 2 8 9 6 1 8 6 5 6 6 7	268.080902	140.102973	139.642598	93.276152	373.021723
13.	9 6 7 1 5 3 5 3 2 2 8 9 10 3 5 6 5 4 1 7	262.257816	144.237973	139.588981	97.477220	381.304175
14.	1 7 3 5 1 1 6 5 10 3 5 4 10 5 3 7 7 8 4 9	256.018838	160.277110	131.579058	98.740089	390.596258
15.	1 10 1 6 8 6 1 10 8 9 1 10 3 9 5 6 3 7 3 1	273.125957	147.440772	139.995484	78.695185	366.131440
16.	10 1 2 8 8 7 4 5 4 9 3 9 1 6 3 10 2 3 1 4	270.642959	147.801842	131.768939	89.919712	369.490492
17.	5 4 10 6 8 7 9 1 8 6 4 3 3 3 6 8 3 9 8 6	288.408810	143.871545	139.729927	63.128575	346.730047
18.	6 8 3 7 5 3 5 9 4 1 2 7 3 8 7 7 3 10 7 6	269.313817	160.655178	131.487603	79.171253	371.314034
19.	9 2 1 2 4 1 6 9 7 5 4 3 6 3 2 9 7 6 9 4	258.571774	152.100713	140.002718	94.636384	386.739815
20.	4 8 1 7 1 8 2 9 10 2 10 1 10 1 1 2 6 3 10 1	273.728648	139.401842	143.494484	82.428973	365.325298
21.	8 2 6 1 2 7 7 7 4 2 7 3 10 8 8 3 10 8 4 6	280.668976	144.136248	131.379058	80.776300	356.291606
22.	4 5 8 8 1 10 3 7 4 8 10 6 4 10 4 10 9 10 10 4	289.436154	139.391248	143.394484	62.713608	345.499339
23.	6 10 10 10 10 3 4 7 4 7 8 4 1 4 5 5 6 1 10 3	267.227817	143.527973	139.527526	91.157040	374.212539
24.	9 2 4 10 4 2 6 7 4 10 7 9 5 5 2 4 4 7 2 8	270.917006	139.756307	139.755364	89.605061	369.116732
25.	4 9 9 1 1 7 8 6 5 5 9 9 7 5 7 2 5 3 5 1	262.763057	152.141307	132.127484	96.302212	380.571003

26.	10 1 1 6 8 6 1 2 8 4 1 10 2 9 3 6 5 9 10 4	266.142809	147.356545	139.929808	88.451773	375.738125
27.	9 2 4 4 10 6 10 7 10 8 1 5 10 1 4 5 3 7 1 5	261.625567	147.562704	135.717819	98.945117	382.225640
28.	9 2 6 8 1 7 9 8 6 9 10 3 10 3 2 7 5 5 10 4	264.013783	152.091545	131.987484	94.689075	378.768104
29.	9 2 6 8 1 6 10 2 10 8 6 9 10 1 4 10 4 7 1 5	271.011537	135.207704	139.996819	93.783458	368.987981
30.	9 2 4 8 1 6 10 2 10 8 1 9 10 1 4 10 10 2 10 5	289.425143	135.202407	135.856364	74.453712	345.512484
31.	**6 5 7 7 1 8 9 2 2 2 5 7 4 10 2 2 9 1 6 5**	**298.563885**	**148.162110**	**139.520436**	**47.254146**	**334.936692**
32.	2 2 9 2 1 1 7 8 4 4 5 3 5 7 1 10 1 4 9 7	269.022956	135.882438	139.490436	96.342616	371.715490
33.	1 8 3 9 2 5 4 10 9 5 6 3 3 9 1 9 10 8 4 4	260.752977	139.941248	143.735939	99.827544	383.504730
34.	9 3 8 5 6 4 7 5 1 7 8 6 1 4 4 2 5 2 4 3	268.960876	143.863270	135.827819	92.110198	371.801287
35.	9 7 3 7 7 9 5 9 6 5 2 3 6 8 7 2 5 3 10 4	277.270805	152.696842	131.827484	76.133913	360.658238
36.	3 9 9 1 7 7 8 6 5 5 6 4 4 3 6 7 3 9 8 6	274.370114	152.321545	139.869927	72.279712	364.471183
37.	3 5 8 8 4 10 3 7 4 2 7 3 10 1 9 1 9 2 6 9	261.402885	144.042407	151.095513	87.413329	382.551249
38.	5 5 5 2 10 2 7 4 4 10 8 1 3 3 8 3 5 8 7 6	263.584204	148.130178	131.217603	100.037624	379.385406
39.	8 9 1 4 8 6 5 8 8 10 7 9 1 1 4 2 10 2 4 4	269.594627	139.366545	135.857939	95.702790	370.927273
40.	9 1 8 5 6 4 7 5 1 7 8 1 1 8 8 2 7 5 8 10	268.126009	147.756545	131.817364	93.385061	372.958970
41.	6 2 6 1 2 7 7 9 4 8 10 6 1 10 2 10 8 8 8 3	284.888727	139.622973	131.927364	79.463903	351.014240
42.	4 9 9 1 1 7 4 6 5 5 6 9 4 5 7 2 4 3 10 6	275.636278	147.831842	136.266603	78.698502	362.796946
43.	5 4 9 5 8 2 3 9 10 10 6 4 7 1 1 9 5 3 8 4	263.167588	143.866842	143.634484	92.484678	379.986003
44.	9 3 1 4 8 1 2 8 7 5 4 10 6 3 4 6 7 5 8 10	261.464521	143.801545	143.624364	95.035159	382.461068
45.	3 3 5 10 9 8 9 2 2 2 5 7 7 10 2 1 9 6 9 7	285.687068	144.312438	143.321598	62.399309	350.033345
46.	6 5 7 10 1 7 1 6 10 3 10 5 4 4 9 1 3 2 10 10	262.371990	147.856545	143.594364	89.687337	381.138246
47.	4 2 9 1 10 9 4 10 5 5 6 3 3 9 1 3 10 3 4 3	265.085992	143.848567	135.957819	97.429693	377.236079
48.	1 8 9 2 10 7 6 8 4 7 5 7 5 7 5 1 5 1 6 5	278.223383	144.007110	135.551436	79.864875	359.423421

49.	9 9 4 4 10 7 2 8 6 9 10 3 10 3 2 4 5 5 8 7	268.895868	143.903270	135.856364	92.131541	371.891175
50.	4 4 2 1 3 10 10 2 7 2 10 7 5 7 3 3 8 8 3 5	265.898606	152.176041	131.307364	92.599799	376.083205

Fitness Sum	Maximum Fitness	Minimum Fitness	Average
1.3559e+004	298.5639	256.0188	271.1784

4.5.2 Cost calculation by simulator

Since there are 20 operations therefore size of chromosome is set to be 20. Chromosome in output of simulator represents the following:

6 5 7 7 1 8 9 2 2 2 5 7 4 10 2 2 9 1 6 5

1st operation is allocated to site 6   13th operation is allocated to site 4 and so on.

Based on the above assumption and Table 3 and Table 4 different costs are calculated for each chromosome as follows:

I/O Cost= I/O Coefficients * Fragment_Size

CPU Cost= CPU Coefficients * Fragment_Size

Communication Cost = Communication Coefficients between two sites * Fragment_Size

Fragment_Size for various operations are already calculated above and I/O Coefficients, CPU Coefficients, Communication Coefficients are taken from Table 3. Some of the cases are shown below:

- Operation 1: Selection   Allocated to site 6

  I/O cost = I/O coefficient for site 6 * Fragment_Size

  = 1*100 = 100

  CPU cost = CPU coefficient for site 6 * Fragment_Size

  = 1.2 *100 = 120

  Communication Cost = 0 as selection does not involve any communication

  Similarly cost is calculated for other selection operation as well.

- Operation 8: Projection   Allocated to site 2

  I/O cost = I/O coefficient for site 2 * Fragment_Size

  = 1.1*70 = 77

  CPU cost = CPU coefficient for site 2 * Fragment_Size

= 1 *70 = 70

Communication Cost = 0 as projection does not involve any communication

Similarly cost is calculated for other selection operation as well.

Only one of the cases is shown below and the best case is taken that is case 3. While calculating the cost for join operator the fragment size used here is taken from case 3 that is combination of joins and inner joins.

- Operation 19: Join $f_{22} \bowtie f_{23}$ Allocated to site 6

  CPU Cost = Site's CPU Coefficient*(f_{22} size * f_{23} size)

  =1.2 *(63*1.0206) = 77.15736

  Total I/O Cost = Site's I/O Coefficient *(f_{22} size * f_{23} size)

  =1*(63*1.0206) = 64.2978

  Communication Cost

  Operation 15 is allocated to site 2, Operation 16 is allocated to site 2. Both are at same site, therefore

  Communication cost = Communication Coefficient between sites 2 and 6 * (f_{22} size * f_{23} size)

  = 14*(63*1.0206) = 900.1692 which is represented as 9.001692 in output due to MATLAB syntax.

  On calculating the communication cost for other operators as well and summing them up gives 47.254146

  In case fragments for join operations are allocated to different sites then for both operations communication costs are calculated and then added together to get the communication cost. In similar manner communication cost for other operations are also calculated.

And finally after calculating the cost for each operation all the costs are added together to get the total cost for each chromosome as shown in the GA output

4.5.3 RESULTS

Simulator is run for different instances of database so as to give image of dynamic database with varied cardinalities each time. It has been observed that when joins and inner joins are used in combinations then communication cost greatly reduces. As shown in GA output the minimum communication cost in different cases is as follows:

Case 1: Using Joins

Minimum communication cost = 72.470890

Case 2: Using Inner joins

Minimum communication cost = 82.502469

Case 3: Using combination of Joins and Inner joins

Minimum communication cost = 47.254146

It shows that when joins and inner joins are used in combination then they are useful. Individually they are incurring more communication cost than when used together.

In the graphs shown below Y1, Y2, Y3, Y4 represent the following:

Y1 = ID⋈ department; Y2 = creditor⋈ student; Y3 = student ⋈ salary; Y4 = student ⋈ salary ⋈ employee

The join operation between ID and department, department_name being the foreign key in ID relation and primary key in department relation so in this case PP_{IJ} (Percentage participation for inner join) will be 1. Also changing the number of rows of these two relations will not change PP_{IJ}. This is shown in Figure 11 and Figure 12. So for the next instances of database this join for relation ID and department has been omitted and in Figure 13 and Figure 14 PP_J and PP_{IJ} has been shown for join operations Y2, Y3 and Y4. Table 5, Table 6, Table 7, Table 8 represent relations with different cardinalities at different instances of database.

department

department_name	department_city	assets
Gaithersburg	Maryland	4025188
Reno	Nevada	5958569
Seattle	Washington	8456156
Gresham	Oregon	9322844
New Orleans	Louisiana	5185219
Houston	Texas	4508766
Birmingham	Alabama	1064468
Salt Lake City	Utah	7386988
Fort Collins	Colorado	7326936
Biloxi	Mississippi	6783510
Southaven	Mississippi	2289090
Wichita	Kansas	6658231
Anchorage	Alaska	8613174
Boise	Idaho	9663276
Tacoma	Washington	4203466
Phoenix	Arizona	6899386
Baton Rouge	Louisiana	7294010
Jonesboro	Arkansas	9679925
Auburn	Maine	2342194
Tulsa	Oklahoma	4796138
Nampa	Idaho	5862492
Bangor	Maine	7008262
Bloomington	Minnesota	8278343
San Jose	California	9813941

Figure 9: Screenshot of database Table department

employee

employee_name	employee_street	employee_city	employee_Pin
Plato	Pocatello	Idaho	79656
Flynn	Bellevue	Nebraska	69991
Gavin	Orlando	Florida	90205
Amal	Bridgeport	Connecticut	89272
Kevyn	Grand Island	Nebraska	63629
Piper	Gary	Indiana	39625
Celeste	Green Bay	Wisconsin	77431
Olympia	Fayetteville	Arkansas	71894
Lewis	West Valley City	Utah	81726
Andrew	Oklahoma City	Oklahoma	48069
Fritz	Kapolei	Hawaii	19444
Ezekiel	St. Petersburg	Florida	17131
Maisie	Iowa City	Iowa	40604
Idona	Springfield	Massachusetts	60561
Mason	Gary	Indiana	18337
Xanthus	Saint Louis	Missouri	42226
Dante	Baltimore	Maryland	16102
Thor	Auburn	Maine	50929
Kiona	Chattanooga	Tennessee	35506
Cleo	Fort Worth	Texas	43670
Levi	Gillette	Wyoming	85278
Lee	Norman	Oklahoma	95922
Nero	Fairbanks	Alaska	99909
Patrick	San Jose	California	93001

Figure 10: Screenshot of Employee Table

Database Instance 1:

Table 5: Database instance 1 .

Tables	ID	Department	Creditor	Student	Salary	Employee
Number of Rows	88	75	70	50	54	110

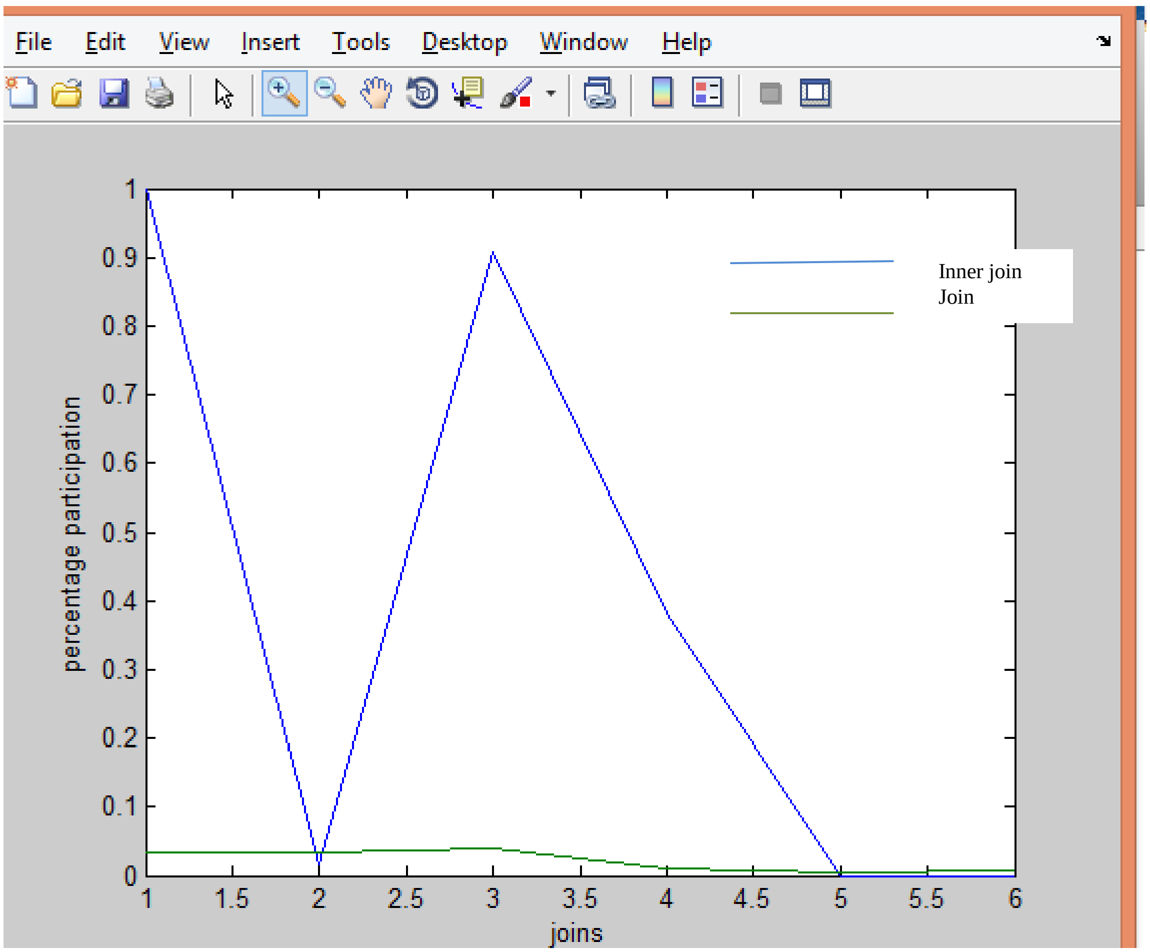


Figure 11: Percentage participations for first database instance.

Database Instance 2:

Table6: Second database instance.

Tables	ID	Department	Creditor	Student	Salary	Employee
Number of Rows	99	80	71	52	55	288

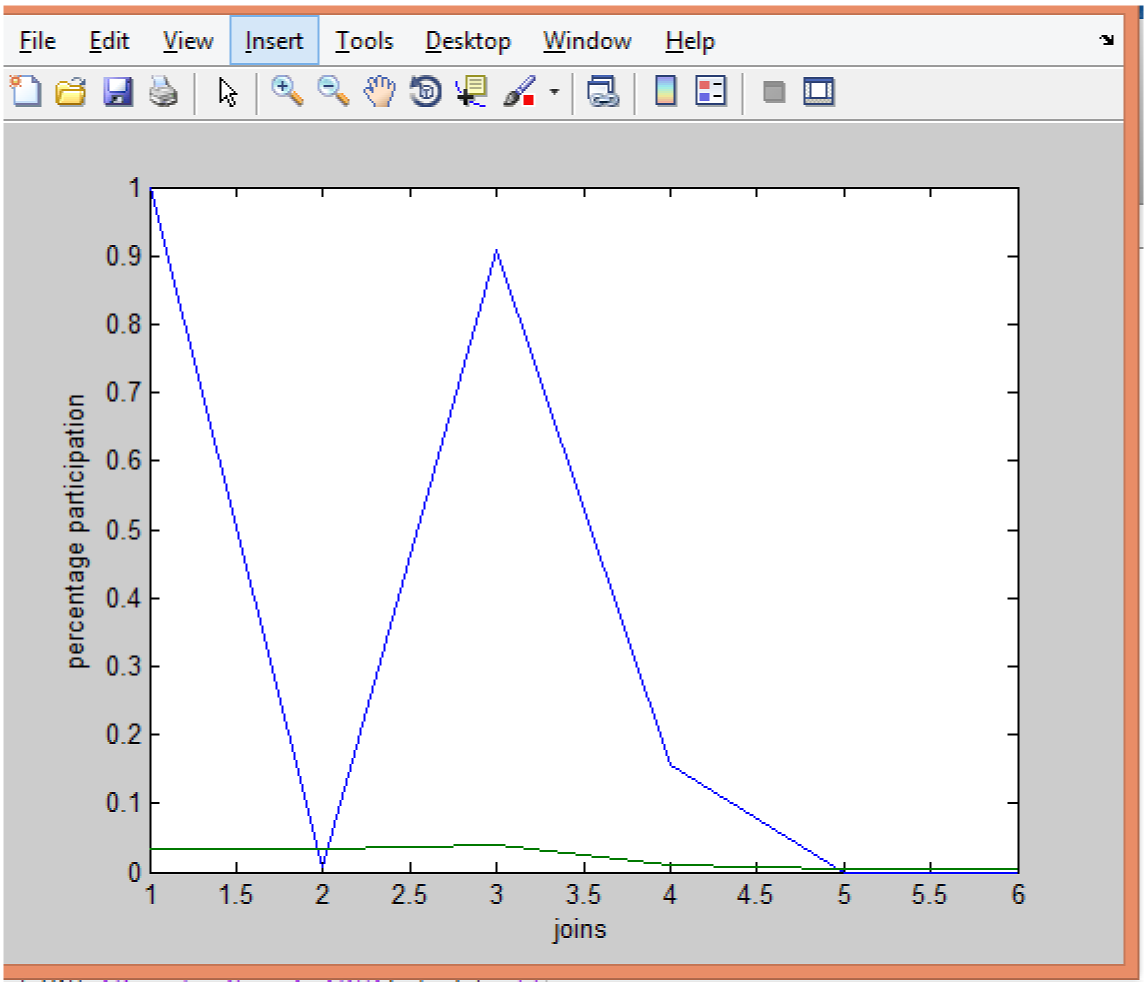


Figure 12: Percentage participations for second database instance.

Database Instance 3:

Table 7: Third database instance.

Tables	ID	Department	Creditor	Student	Salary	Employee
Number of Rows	98	81	69	11	80	298

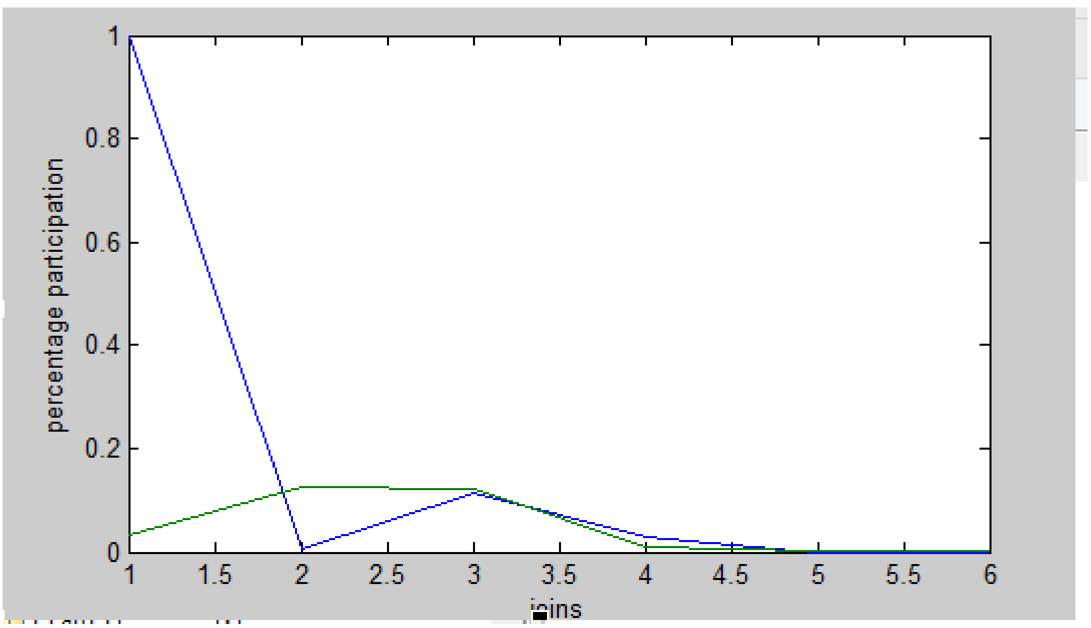


Figure 13: **Percentage participations for third database instance.**

Database Instance 4:

Table 8: Fourth database instance.

Tables	ID	Department	Creditor	Student	Salary	Employee
Number of Rows	52	81	70	5	98	203

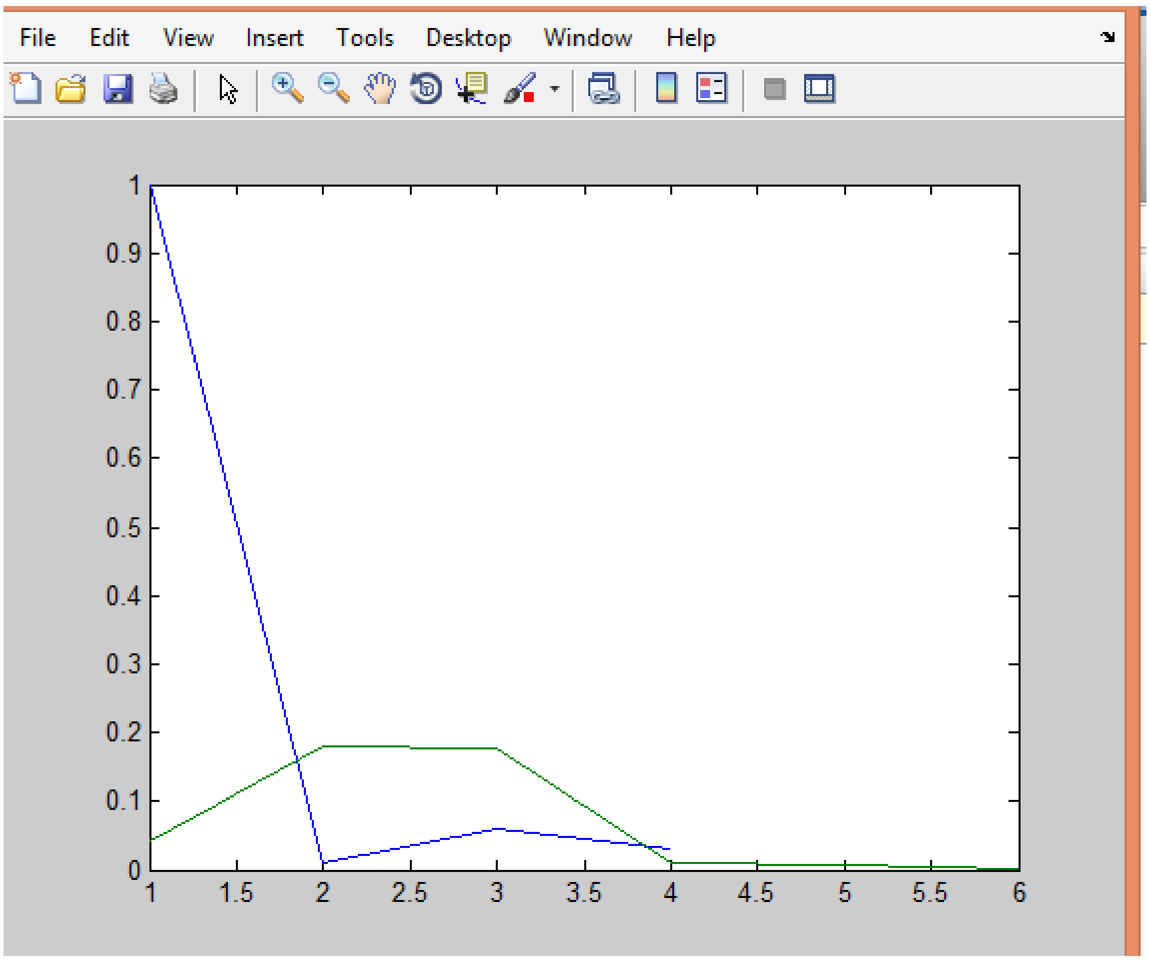


Figure 14: Percentage participations for fourth database instance.

Figure 15 denotes the percentage reduction in communication cost wherever inner join is beneficial. In this graph first bar represent that only at one place inner join was beneficial for first database instance and percentage improvement is written above the bar. Similar is the case for second database instance. For the third and fourth instance at two places it is proving to be beneficial than joins. Although in very less cases inner join is coming out to be beneficial but percentage reduction in communication cost is coming as high as 90% in those cases. So inner join greatly reduces the communication cost involved but in very few cases.

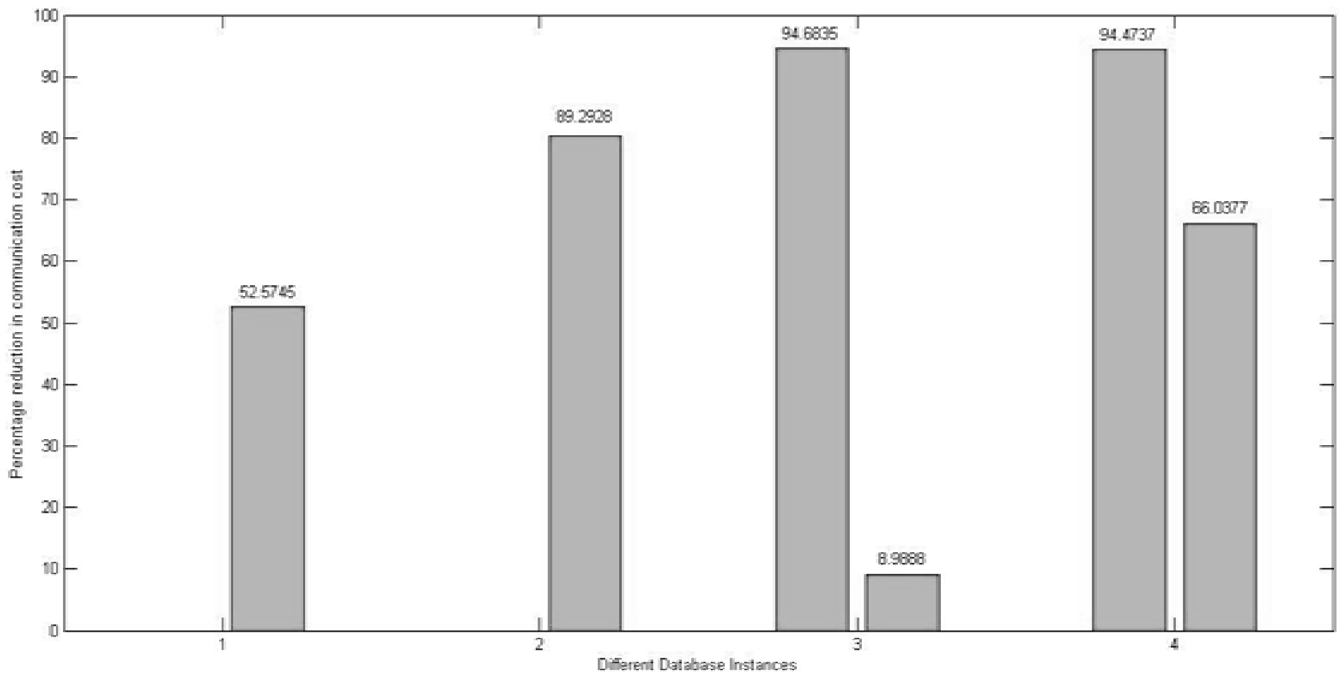


Figure 15: Percentage reduction in communication cost for inner join operations.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

Query processing in a distributed database requires transfer of data from one computer to another through a communication network. Query at a given site might require data from remote sites. In query optimization, a cost is associated with each query execution plan. Cost is the sum of local cost (I/O cost, CPU cost at each site) and the cost of transferring data between sites. The complexity and cost increases with the increasing number of relations in the query. A query execution strategy or plan is required to minimize the cost of query processing.

The cost of processing a DD query is the entire cost measure. The entire cost size is the sum of all cost components. On executing join and inner join, communication costs among various sites may be incurred along with the local processing cost.

After experimenting with the actual dynamic database on calculating the selectivity factors dynamically it is seen that in very less cases $PP_{IJ}<PP_{J}$. Whenever selectivity factor for inner join comes less than join then only inner join should be used otherwise join should be used. Whenever very few tuples are required which is a very rare case for join operation only then inner join should be used otherwise normal join should be used.

From the results it has been found that neither using only joins incur minimum communication cost nor using inner joins alone reduces communication cost. Rather when they are used in combination that is somewhere joins are used and somewhere inner joins are used then communication cost greatly reduces. So in very less cases inner join has proven to be beneficial. It has been observed that when very few tuples are required for joining the relation at other site only then inner join should be used.

Future work

Cost analysis of the distributed query can be further studied by using Genetic algorithm approach that gives optimal results within small time interval. Enumerative and Deterministic procedures are designed to find the best solution but they go almost intractable as soon as the number of sites or number of complex operations like joins are increased to double digits or more. Whereas a genetic solution does not guarantee the finding of an optimal solution but can provide a very good solution in an extremely small time as compared to deterministic one. In future efforts should be done to incorporate Genetic Based Solutions to allocation problems of Distributed Database. More work needed to be done to ensure that an optimal solution is guaranteed in most of situations by GA.

REFERENCES

[1] Chhanda Ray, (2009) *Distributed Database Systems*, Publisher: Pearson Education India Publications, New Delhi.

[2] Korth, Henry F., Abraham Silberschatz, and Henry F. Korth, (1986) *Database system concepts*. Vol. 582. New York: McGraw-Hill.

[3]Baiao, Fernanda, Marta Mattoso, and Gerson Zaverucha. (2000) "Horizontal fragmentation in object dbms: New issues and performance evaluation."*Performance, Computing, and Communications Conference, 2000. IPCCC'00. Conference Proceeding of the IEEE International*. IEEE.

[4]Bamnote, G. R., and S. S. Agrawal. (2013)"Introduction to Query Processing and Optimization." *International Journal* 3.7.

[4] Rho Sangkyu, T. March Salvatore, (2002) "A Comparison of Distributed Database Design Models", Seoul Journal of Business Vol. 8 No. 1.

[5] Rho Sangkyu, T. March Salvatore, (1994) "A Nested Genetic Algorithm for Distributed Database Design", IEEE.

[6]Bertino, Elisa. (1992) "A view mechanism for object-oriented databases." *Advances in Database Technology—EDBT'92*. Springer Berlin Heidelberg.

[7] Ceri, Stefano, Barbara Pernici, and Gio Wiederhold, (1987) "Distributed database design methodologies." *Proceedings of the IEEE* 75.5,p 533-546.

[8] Connolly, Thomas M. (2005) *Database systems: a practical approach to design, implementation, and management*. Pearson Education.

[9]Councill, Bill, and George T. Heineman. (2001) "Definition of a software component and its elements." *Component-based software engineering: putting the pieces together*: 5-19.

[10] B.M. Monjurul Alom, Frans Henskens and Michael Hannaford, (2009) "Query Processing and Optimization in Distributed Database Systems", IJCSNS Vol. 9 No. 9.

[11] Peter Scheuermann, Eugene Inseok Chong, (1997) "Adaptive Algorithms for Join Processing in Distributed Database Systems".

[12] Manik Sharma, Gurdev Singh, Rajinder Virk, (2012) “Analysis of Joins and Semi Joins in a Distributed Database Query”, IJCA.

[13] Hevner, Alan R., and S. Bing Yao, (1979) "Query processing in distributed database system." *Software Engineering, IEEE Transactions on* 3: 177-187.

[14] Xiaofeng Li, Dong Le, Hong Zhi Gao, Lu Yao, (2010) “Study of Query of Distributed Database Based on Relation Semi Join”, International Conference On Computer Design And Appliations (ICCDA).

[15] Jim Wilenius, “Randomized Algorithms and Heuristics for Join Ordering”, (2007) Computing Science Division, Dept. of Information Technology, Uppsala University.

[16] John, Rajan, and V. Saravanan., (2008) "Vertical Partitioning in Object Oriented Databases Using Intelligent Agents." *IJCSNS* 8.10: 205.

[17] Carolyn Mitchell, (2004) “Components of a Distributed Database”, Department of Computer Science, Norfolk State University.

[18] Zhou Lin, Chen Yan, Li Taoying, Yu Yingying, (2012) “The Semi-join Query Optimization in Distributed Database System”, National Conference on Information Technology and Computer Science (CITCS).

[19] Kunii, Hideko S. "Data Manipulation Language." *Graph Data Model*. Springer Japan, 1990. 29-39.

[20] W. Cornell Douglas, S. Yu Philip, 1989 “On Optimal Site Assignment for Relations in the Distributed Database Environment”, IEEE.

[21] Shahidul Islam Khan and Dr. A. S. M. Latiful Hoque, (2010) “A New Technique for Database Fragmentation in Distributed Systems”, IJCA Vol. 5 No. 9.

[22] Ray, Chhanda, (2009) *Distributed Database Systems*. Pearson Education India,.

[23] Narasimhaiah Gorla and Suk-Kyu Song, (2010) “Subquery Allocations in Distributed Databases Using Genetic Algorithms”, JCS&T Vol. 10 No.1.

[24]Robbins, Robert J, (1994) "Database Fundamentals." *Johns Hopkins University, rrobbins@ gdb. org*.

[25] Rupley, M. L, (2008) "Introduction to query processing and optimization." *Indiana University at South Bend* .

[26] Sharma, Manik, et al. (2013) "Stochastic Analysis of DSS Queries for a Distributed Database Design." *International Journal of Computer Applications* 83.5: 36-42.

[27] ŞTEFAN, Ileana, and Maricel POPA, (2008) "Distributed Database Design–Top-Down Design".

[28] Tamer èOzsu, M., and Patrick Valduriez. (2011) *Principles of distributed database systems*. Springer.

[29] R.S. Virk, (2012) “Optimized Access Strategies for a Distributed Database Design”, Ph.D dissertation, Dept. of Comp. Sci. & Eng., Guru Nanak Dev Univ., Amritsar.

[30]Mitchell, Melaine, (1999) “An introduction to Genetic Algorithms”, 5th ed. England, The MIT Press Cambridge, Massachusetts.

[31] Wells, Garth, (2001) "Data Definition Language." *Code Centric: T-SQL Programming with Stored Procedures and Triggers*. Apress, 2001. 35-70.

[32] Gen. Mitsuo, Cheng. Runwei, “Genetic Algorithms and Engineering Optimization”, 3rd ed., New York, A Wiley-Interscience publication, 2000

[33] Khurana, Namita et al., (2011) “Genetic Algorithm: A Search of Complex Spaces”, in International Journal of Computer Application, Vol. 25.

[34] Tom V. Mathew, (2008) “Genetic Algorithm”, Dept. of Civil Eng., Indian Institute of Technology, Mumbai.

[35] Donald Kossman, 2000 “The State of the Art in Distributed Query Processing”, ACM Computing Surveys.

[36] T. March Salvatore and Rho Sangkyu, (1995) “Allocating Data and Operations to Nodes in Distributed Database Design”, IEEE.

[37] Fan Yuanyuan, Mi Xifeng, (2010) “Distributed Database System Query Optimization Algorithm Research”, IEEE.

[38] Richard L. Cole, (1994) “Optimization of Dynamic Query Evaluation Plans”, ACM.

[39] Whitley, Darrell. (1994) "A genetic algorithm tutorial." *Statistics and computing* 4.2: 65-85.

[40] Sharma, Manik, et al. (2013)"Stochastic Analysis of DSS Queries for a Distributed Database Design." *International Journal of Computer Applications* 83.5 :36-42.

[41] Mamaghani, Ali Safari, et al. (2010) "A novel evolutionary algorithm for solving static data allocation problem in distributed database systems." *Network Applications Protocols and Services (NETAPPS), 2010 Second International Conference on*. IEEE.

APPENDIX

Glossary of Terms

- Crossover: It is a genetic operator that combines (mates) two chromosomes (parents) to produce a new chromosome (offspring).

- Chromosome: It consists of "genes" (e.g., bits), each gene being an instance of a particular "allele".

- Data allocation: It is the prominent activity in the distributed database which decides that where to locate the data.

- Database Management System: It is used to manage whole data in organizations.

- Distributed database: It is a collection of logically interrelated databases that can be stored at different computer network sites.

- Data definition language (DDL): Users can specify structure of database, data types and constraints on data by using DDL.

- DML: It provides general facility to enquire about data, which is known as query language.

- Fragmentation: The process of dividing the relation into sub relation is called fragmentation.

- Genetic Algorithm: GA was a method for moving from one population of "chromosomes" to a new population by using a kind of "natural selection" together with the genetics–inspired operators of crossover, mutation, and inversion.

- Horizontal Fragmentation: It means to divide a relation along its rows.

- LPC: Local Processing Costs for processing a query's simple selection & projections may be represented as costs of transforming input relation from disk to memory and CPU time for processing a selection or projection at site

- Mutation: It is a genetic operator that alters one or more gene values in a chromosome from its initial state.

- Query: Database Query is a way of instructing DBMS to update, insert, retrieve, and delete data from database.

- Query optimization: It is the function of determining the most efficient query plan among all, which is performed by query optimizer.

- Replication: It consists of keeping of copies of complete database at each site.

- Reproduction: Reproduction selects good strings from the population and puts them in mating pool selection operator chooses those chromosomes in the population that will be allowed to reproduce, and on average the fitter chromosomes produce more offspring than the less fit ones.

- Vertical Fragmentation: Vertical fragmentation partitions a relation long its attributes.

Abbreviations

- A_{rs} : Data Allocation Variable
- S^{q}_{ys} : Sequence of various sites where sub queries gets executed
- LPO: Left previous operation of a join
- RPO: Right previous operation of a join)
- LPC: stands for Local Processing Cost
- CC: Communication Cost
- : No. of memory blocks of relations ‘r’ accessed by sub query ‘y’ of q.
- IOC_s : Input Output Cost Coefficient of site s in millisecond per 8k bytes
- CPC_s: CPU Cost coefficient of site s.
- ρ_p : Percentage Participation
- : It is the size of an intermediate relation.
- : It is the communication cost coefficient between site s and v