2011

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

Proceedings

https://csbapp.uncw.edu/mscsis

OPTIMIZING JOIN QUERY IN DISTRIBUTED DATABASE

Shun Jiang

A Capstone Project (or Thesis) Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science
Department of Information Systems and Operations Management

University of North Carolina Wilmington

2011

Approved by

Advisory Committee

Dr. Bryan Reinicke | Dr. Devon Simmonds | Dr. Ulku Yaylacicegi

Dr. Clayton Ferner, Chair

Accepted By

Dean, Graduate School

Abstract

This project proposes an algorithm which attempts to predict the best execution plan for a join query retrieving data from two remote computers. The goal of this algorithm is to execute efficiently the join query in a distributed database. My algorithm estimates the response time for the possible execution plan based on the size of the transmission data, the transmission speed, and the machine processing speed. Also, the report presents a service-oriented system which executes a join query using my algorithm. The report describes the system development activities under the Unified Process model.

This project conducts a test on data transmission. The test result shows that the total response time of data transmission in my system is comprised of the time of querying, transferring, and inserting. The times of querying, transferring and inserting are linear and have a strong correlation with the amount of transmission data. The parallel query process can save the time of executing the SQL query which retrieves data from remote sites. A test of join query performance indicates that the size of transmission data and the transmission speed have impacts on the total response time. By comparing the estimated data size and estimated response time with the actual execution results, I discovered that my algorithm can predict the best plan for most of the join queries with an exception of join queries selecting only a small amount of data.

Table of Contents

Chapter 1: Introduction

Data is becoming more and more important in the current computing environment and enterprise application. As the volume of data is increasing, the databases based on one centralized machine suffer from either storage limitations or computing bottlenecks [1]. A distributed database is more popular because it improves system performance, reliability, availability and modularity that are inherent in distributed systems

However, join query execution is more complex in a distributed database than in a centralized database. In a distributed database, joining data on different machines requires the transmission of data. This data transmission could be time-consuming if the quantity of transmission data is large. 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 currently used in the distributed database systems. 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. Therefore, these two methods are not good for all types of join query.

In this project, I will present an algorithm which attempts to identify the best execution plan for a join query that retrieves two tables from different remote sites. My algorithm predicts the best execution plan by estimating and comparing the time cost of different possible execution plans. Also, I will propose a service-oriented architecture system which can perform the join query in distributed environment using my proposed algorithm. The system development activities are implemented by following the Rational

Unified Process Model. At the end, my system and algorithm will be evaluated by measuring the response time of join query execution.

The rest of my capstone project report is organized as follows: Chapter 2 introduces the distributed database system benefits and the join query performance, and reviews the two join query execution methods. Chapter 3 represents my algorithms and system architecture. Chapter 4 describes all my system development activities under the Rational Unified Process Model. Chapter 5 shows results and findings of system tests.

Chapter 2: Background

2.1 Distributed Database

Distributed database (DDB) is defined as a collection of multiple, logically interrelated databases distributed over a computer network [2]. The databases within the distributed database could be distributed not only across the local area network inside one company but also across several companies through a wide area network. As with a traditional centralized database, distributed database systems must hide the heterogeneity of the data resources and provide a unified way to access and manage those data resources. *Distributed database management system* (DDBS) provides the functions to manage the distributed database and makes the distribution transparent to users. It helps users to manipulate the entire distributed database without considering the underlying data distribution details [3]. Figure 1 shows the architecture of the distributed database.

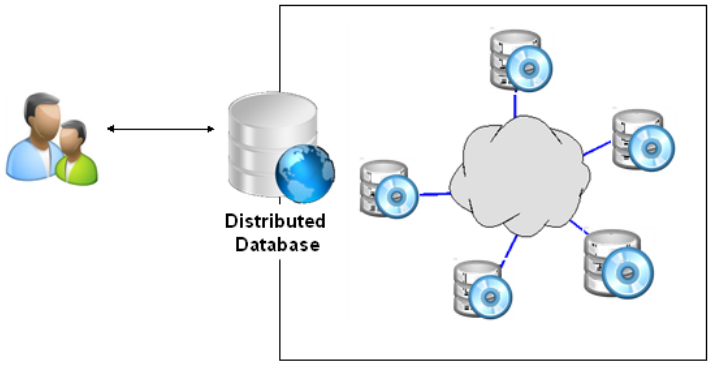


Figure 1- Overview of Distributed Database System

2.2 Benefits of Using Distributed Database

Compared to the traditional centralized database, the distributed database has the following advantages:

1) Local autonomy: Local autonomy means that all the data in the distributed database is owned and managed locally. Branches, departments and users in an enterprise or different companies do not rely on the single centralized database, but design, implement and manipulate their own databases to satisfy their needs and fulfill their job responsibilities. For example, the Human Resources department has the HR database which only stores the employee information rather than other departments' information. Users can deal with the data which they familiar with so that the operation efficiency is enhanced. Furthermore, each site has the ability to control local data, administer security, log transactions, backup and recover policy. The authority is delegated to each site administrator. Therefore, the database management is simplified and the responsibility and accountability are clear. As a result of the local autonomy, the distributed database reflects the organizational structure.

2) Improved Performance: Distributed Database has three potential advantages on performance:

a. Since each site handles only a portion of the database, contention for CPU and I/O services is not as severe as for centralized databases [2]. In the central database architecture, one single machine handles all the transactions and is shared with all the users. In the Distributed Database, the data reside on different machines and only a group of users share one machine resource simultaneously.

b. Localization reduces remote access delays that are usually involved in wide area networks. Data are stores at or close to the location where they are needed most. Users are able to access the data via high speed local area networks. Thus, users spend less time on data transmission.

c. Database responses fast due to small size. Each database stores the data and handles the functions needed by one branch or department. The volume of data on each database is small and the data structure and database design are simple, so the distributed database can completes the queries faster than central database which has huge volume of data and complicated data structure.

3) Reliability: Distributed Database system are intended to improve reliability since they have replicated components and, thereby eliminate single points of failure [2]. Company can have several copies of important data on the computers distributed on different locations. If one computer crashes or one location has an emergency event, the distributed database systems can retrieve and restore data from other computers or locations.

4) Availability: In the distributed database system, it is not sufficient to bring down the entire system that one or more sites are not available. On one hand, the data or functions on the sites where are unreachable can have a replication on other sites if the distributed database administrator design the system carefully. The system can use the data on other site to keep the system is operating. On the other hand, although data and functions on one or some databases are lost, the rest of the system is still available. For example, the accounting department can keep doing most of its jobs when the database in human resource is down. In the central database system, all the systems are down when the single central sever crashes, so business operation is interrupted until the central server is restored.

5) Easier System Expansion: The capacity of the distributed database can be increased easily by adding the computers to the network. The expansion can have no negative impact to the running system. Another reason for easier system expansion is

economic cost. In the distributed environment, system can associate and coordinate a number of small machines to gain the power equivalent power of a supercomputer. Small machine is much cheaper than supercomputer, and doesn't require an expensive facility for installation and protection.

2.3 Join Query in a Distributed Database

Although a distributed database system has the advantages of computation ability, reliability, availability and flexibility, it has issues on database interaction, design and security. One database interaction issue is the join query execution across sites.

A *normalized* database is one where the data has been broken out from larger tables into many smaller tables for the purpose of eliminating repeating data, saving space, improving performance, and increasing data integrity [5]. For example, the department information is stored in the department table, and employee table simply refers to the department table. Figure 2 is the diagram of department table and employee table.

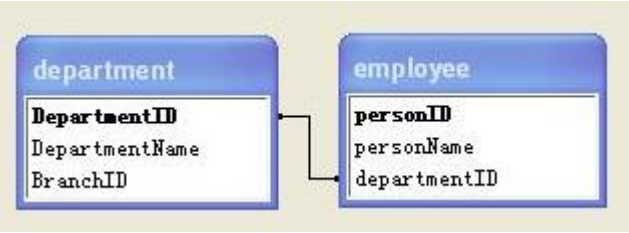


Figure 2 - Diagram of department and employee

A *Join query* defines as how to take and combine multiple tables into one resulting set [4]. In this normalized environment, we frequently need information which is not from one table but several tables. If we want to know each employee and in which department the employee works, we need to retrieve the data from both Department and Employee table. For example, we can run the following join query:

SELECT * FROM employee e

JOIN department d ON e.departmentID = d.departmentID

It is more complex and time-consuming to execute this join query 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. A straightforward 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 [6] as depicted in figure 3. This requires movement of much data between sites, and the time spent on data transmission is usually significant to the entire query response time. As a result, it usually takes a longer time to complete the join query in a distributed database than in a centralized database.

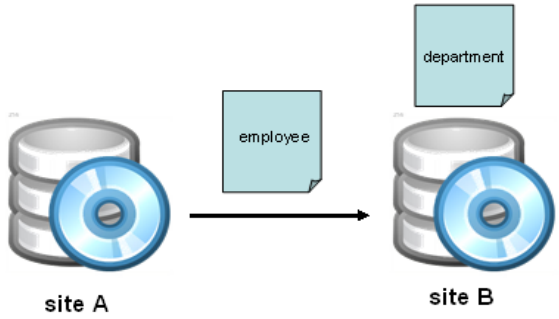


Figure 3 - Join Query Execution on Distributed Database

The goal of *join query optimization* is to execute a join query in a DDS as efficiently as possible in order to minimize the response time [3]. A join query can be executed in several different query execution plans which results in transfer of data between sites via different path or in different orders. Paths have different transmission speed, and the execute order can affect the quantity of data transferred. The transmission speed and the quantity of data transferred are the main factors that impact the query response time. Besides the time spent on data transmission, the time spent on joining data

has considerable impact on the total query response time. The time spent on joining data depends on the machine computation ability. The join query optimization must be able to find out the best or close to the best execution plan which nay require transferring less data faster and joining data on a more powerful machine.

2.4 Related Work on Optimizing Join Query Performance in DDB

In recent research on join query optimization, there are two approaches for join query execution: data size analysis and parallel processing.

2.4.1 Data Size Analysis

[3] introduces a join optimization method which minimizes the quantity of transmission data between two join participating sites. The quantity of data has considerable impact on the join query performance for a distributed database system. Therefore, it is very important to run the query with the least transmission data. The basic strategy of data analysis is to send the smaller table to the site that contains the larger table, and perform the join on that site. Figure 4 illustrates how to apply this strategy to the join query when two tables are stored on different sites are involved.

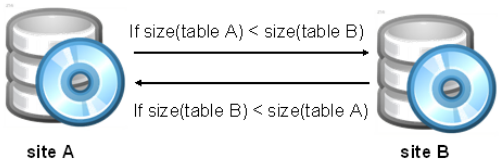


Figure 4 - Minimize the Quantity of Transferred Data

2.4.2 Parallel Processing

[4] presents the parallel query processing for the join query in a distributed database. Parallel processing doesn't focus on minimizing the quantity of transmission data but rather maximizing the number of simultaneous transmissions.

Figures 5 shows a query where site C requires the result of the join query based on two tables from two other sites. In the sequential processing, the network resource between site B and site C is not utilized during the time when site A transfers data to site B and site B performs the join query. In the parallel processing, site A and site B send data to site C simultaneously, and then the join query is executed on site C. As a result, parallel processing can reduce the time spent on transfer data by using more network resources, but it may not transfer the minimum quantity of data needed for the result. The transfer may not actually be simultaneous, but this research will determine whether or not it is.

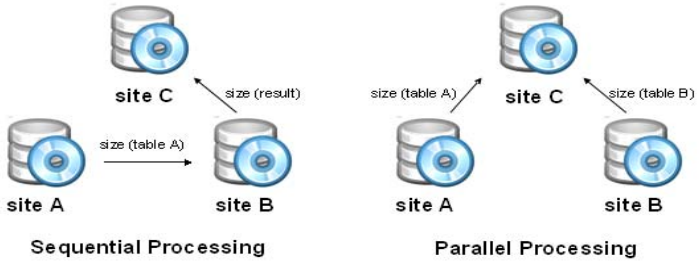


Figure 5 - Data Analysis and Parallel Processing

2.4.3 Limitation

To determine whether data analysis or parallel processing is the best join query execution method, I am going to discuss the response time for each method. We will ignore the other costs on local site, so the query response time is only affected by data transmission. The estimated response times by using the data analysis and parallel processing are as followings:

RESPONSE TIME_ data analysis = TIME (min (size A, size B)) + TIME (size (result))

RESPONSE TIME_ parallel processing = TIME (max (size A, size B))

, where TIME(x) is the time required to transfer x bytes between sites. Also, assuming the data transmission speeds among sites are the same, the transmission time

totally depends on the quantity of transmission data. The response times of those two methods are below:

RESPONSE TIME_ data analysis = MIN (size A, size B) + size (Result)

RESPONSE TIME_ parallel processing = MAX (size A, size B)

When MIN + Result < MAX → Result < MAX - MIN, data analysis is the better method. In this situation, the difference in the sizes of the two participating tables is large relative to the size of result. The parallel processing is better than data analysis if the sizes of two tables are similar or the size of result is large. As discussed above, data analysis and parallel processing can not be the best execution plan for all types of join queries. Therefore, the sophisticated algorithm should consider both methods and determine which one is best for the specific join query.

Also, the response time for parallel processing is probably longer than the time for transferring the maximum size of the two tables. The machine which executes the parallel processing must have multiple CPUs to run jobs simultaneously. If the machine has only one CPU, the parallel transmission will turn to sequential transmission, and the response time will be the sum of time for transferring two tables in sequential. In addition, the inbound network bandwidth of the destination machine must be greater or equal to the sum of outbound bandwidth of two source machine. If the bandwidth is lower, the time for data transmission will be longer than the estimated response time.

Furthermore, although the quantity of transmission data is an important factor, there are other factors which also impact the join query performance like transmission speed and server processing time. If the optimization algorithm ignores these factors, the solution will not provide the best response time. For example, a join query execution plan may have the least quantity of transmission data but transmits data through the slowest

network path or joins data on a slow computer, so the response time of this plan is still long. Therefore, it is necessary for the optimization algorithm to consider all the factors.

2.5 Web Services and Distributed Database

Service-oriented architecture (SOA) is essentially a suite of interoperable services that can be used within multiple separate systems from several business domains. It is gaining popularity as an appropriate system engineering approach for enabling distributed, heterogeneous software components to communicate and interoperate through web services [7].

2.5.1 Standards of Web Services

Web service is an interface that describes a collection of operations that are network-accessible through standardized XML messaging [8]. Web services which architecture is shown in figure 6 have three basic standards:

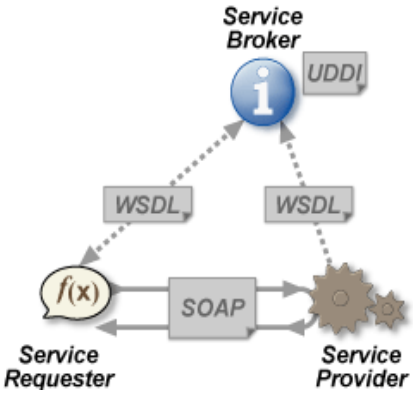


Figure 6 - Web Services Architecture

- *Web Services Description Language (WSDL)*: is an XML-based language that provides a model for describing Web services. The Web service architecture relies on WSDL to specify service functional aspects [9].
- *Universal Description Discovery & Integration (UDDI*): is a service registry that supports publication and discovery of service providers. Service

specifications offer sufficient information and this information is published in UDDI [10].

- *Simple Object Access Protocol (SOAP):* is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks.

2.5.2 Web Services and Join Query Execution in Distributed Database

Join query processing needs to manipulate data on different store devices across the network. Web services support interoperable machine-to-machine interaction over the network. All services can be accessed and requested by a remote site via the network. So, the join query processing system can be developed as a set of web services, and machines in the distributed database can interoperate with each other by calling services on related machines.

Moreover, the distributed database system must be platform-independent. The system must be able to run on heterogeneous operating systems, database platforms, network configurations, and hardware architectures and so on. Web services are platform-independent and portable, because they are based on XML, HTTP and SOAP, which are standard protocols. Furthermore, web services just define the application interface not the underlying implementation. Therefore, the distributed database system based on web services can deal with the heterogeneous database platform environment.

Also, the distributed database system must be able to flexibly change the number of databases and computers. New computers can be added into the distributed database system without changing the configuration on the other computers. Because web services are loosely coupling with each other, it is easy to add, remove, combine and reuse those

services. The web services on the new computers can integrate into the legacy system without any impacts or interruptions.

2.6 Summary

Distributed database system improves performance, reliability, availability and modularity that are inherent in distributed systems. However, Distributed database system has a performance issue on joining tables which are stored on different remote sites. There are two recent join query optimization methods, data analysis and parallel processing. These two methods cannot execute all the join queries in a distributed database efficiently. To improve the query performance, an algorithm should consider these two methods and determine the best query execution plan for specific join query. In addition, the algorithm can be implemented as web services which support machine-to-machine interaction via network, platform independence, and scalability.

Chapter 3: System Analysis and Design

3.1 Join query optimization algorithm

My join query optimization algorithm attempts to find the best execution plan for a join query which accesses data on two remote sites by considering the impact of data size, transmission speed, and server process speed. This algorithm calculates the response time for the possible execution plans in both sequential and parallel way. Finally, this algorithm executes the plan which has the minimum estimated response time. The join query optimization process using my algorithms is shown in Figure. 7.

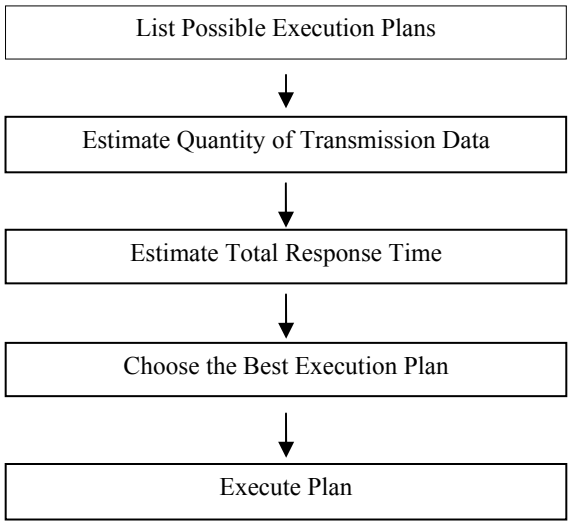


Figure 7 - Join Query Optimization Process

3.1.1 Possible Execution Plan

The first step of my algorithm lists all the possible execution plans. Since three sites participate in the whole process, there are three possible execution plans. Two plans transfer data sequentially and one transfers data in parallel. For example, site C (local site) needs the join query result and site A and site B store the data required by the join query. The three possible execution plans are following:

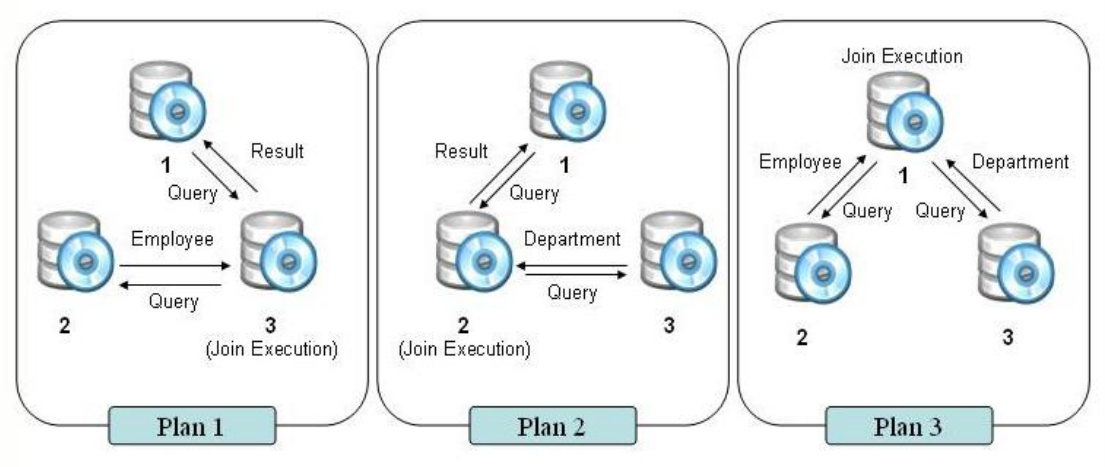


Figure 8 **- Three Possible Execution Plan**

3.1.2 Estimate the Quantity of Transmission Data

In this stage, my algorithm estimates the quantity of transmission data in each step. In those three possible plans that discussed in 3.1.1, we need to know the quantities of three query results: the quantities of data from two source tables, and the quantity of join query result. My algorithm divides one join query into three sub queries and estimates the sizes of those sub-queries results. Here is an example:

The following join query, **SELECT * FROM employee e JOIN department d ON e.departmentID = d.departmentID**, can be divided into these three queries:

A. **SELECT * FROM employee**
B. **SELECT * FROM department**
C. **SELECT * FROM employee e JOIN department d ON e.departmentID = d.departmentID**

Definition 1: Card (A) is the number of rows in table A. For example, "Select count (*) from A"

Queries A and B select the data from two source tables. Because this data resides on the remote machines, the executions of these two queries do not require data transmission. My system can retrieves the sizes of these two queries using this formula,

Size = Card (A) * row size. The row size is the average size of a certain number of rows. My algorithm counts the total size of the first 50,000 rows in the table and calculates the average size by dividing the total size by 50,000.

However, query C is the join query which can not be executed until the data on the remote sites have been transferred to the same sites. Therefore, my algorithm estimates the quantity of join query result based on the join query type and the results of query A and B.

In SQL language, there are five main types of join queries: *inner join*, *left outer join*, *right outer join*, *cross join* and *full outer join*.

Definition 2: Card (A.*field* = *X*) is the number of rows in table A where field *field* is equal to *X*, for example, "SELECT count (*) from A where A.*field* = *X*".

Definition 3: Card (A.*field* != *X*) is the number of rows in table A where field *field* is not equal to *X*, for example, "SELECT count (*) from A where A.*field* = *X*", for example, "SELECT count (*) from A where A.*field* <>*X*".

Definition 4: set (A.*field*) is the set of value for field *field* in table A, for example, "select distinct (*field*) from A"

Inner join returns only the records from two tables where there are matches for whatever field is used for the JOIN. If there are 3 rows in table A and 4 rows in tables which have the same value on the join field, the result of joining those rows will contain 3 * 4 = 12. That means the result of an *inner join* is the Cartesian product of matched records in two tables.

Definition 5: Card (A inner join B on A.*field* = B.*field*) =

$$\sum_{X} \text{Card}(\text{A.field} = \text{X}) * \text{Card}(\text{B.field} = \text{X}) \ \forall X \in set(A.field) \cap set(B.field)$$

Join query has the concept of sides, a left and a right. The first named table is considered as being on the left, and the second named table is considered to be the right. For example, in this join query, *SELECT * FROM employee e JOIN department d ON e.departmentID = d.departmentID*, employee table is on the left side, and department table is on the right side.

The result of a *left outer join* (or simply left join) for two tables always contains all records of the left table, even if the join-condition does not find any matching record in the right table. If there are zero records in right table that match the join field, the join will still return a row but with NULL in each column as the result from right table. Therefore, a left outer join returns all the records from left table, plus the matched records or NULL from the right table.

Definition 6: Card (A left outer join B on A.*field* = B.*field*) = Card (A inner join B on A.*field* = B.*field*) + $\sum_{X} \text{Card}\,(\text{A.field} = \text{X}) \;\; \forall X \in set(A.field) \cap X \notin set(B.field)$

Right outer join has the similar concept to *left outer join*, except its action on reserved table. A right outer join returns all the records from right table, plus the matched records or NULL from the left table.

Definition 7: Card (A right outer join B on A.*field* = B.*field*) = Card (A inner join B on A.*field* = B.*field*) + $\sum_{X} \text{Card}\,(\text{B.field} = \text{X}) \;\; \forall X \notin set(A.field) \cap X \in set(B.field)$

Full outer join combines the effects of both left and right outer joins. It returns the Cartesian product of matched records, and also a row which contains a record from one side, and NULL values for each column from another side if there is no matched record.

Definition 8: Card (A full outer join B on A.*field* = B.*field*) = Card (A left outer join B on A.*field* = B.*field*) + Card (A right outer join B on A.*field* = B.*field*) - Card (A inner join B on A.*field* = B.*field*)

Cross join combines one row in one table to all the rows in another table no matter their join fields are matched or not. The result of cross join is the Cartesian product of two tables. Here is a table shows the result for each type of join query.

Definition 9: Card (A cross join B) = Card (A) * Card (B)

First, my algorithm calculates the number of rows return from the join query using the relationship between join query type and result size. My algorithm detects what type of join query is being performed. Then, my algorithm retrieves the data distribution information for the two participating tables by running this query, "SELECT join_field, COUNT (*) FROM source_table GROUP BY join_field". This SQL query returns the number of rows for each value on the join field in the source table. Table 1 is the example of the data distribution information of the employee table. This distribution information shows there are 10 employees are in department 1 and 20 employees are in department 2. After my algorithm knows the join query type and data distribution information, my algorithm can calculates the number of rows in join query result according to the definitions 5, 6, 7, 8 and 9.

Department ID	Number of Employee
1	10
2	20

Table 1 **– Example of Distribution Information**

Next, my algorithm determines the number of columns and estimates the size of columns in the two join participating tables. The number and size of column can be

retrieved from query result metadata which is provided by JDBC. My algorithm calculates the quantity of each row in each table by multiplying the number and the size of the columns. Also, the number of column is used to calculate the quantity for NULL records that result from the *left outer join*, *right outer join* and *full outer join*. The quantity for the NULL records is equal to the count of NULL times the number of columns. Once my algorithm has the information on row and column of the two participating tables, the size of join query result can be estimated according to Table 2.

Join Type	Total Quantity
Inner join (A inner join B on A.*field* = B.*field*)	*Card (A inner join B on A.field = B.field)* (Row_Size(A) +Row_Size (B))*
Left out join (A left outer join B on A.*field* = B.*field*)	*Card (A inner join B on A.field = B.field)*(Row_Size(A) +Row_Size (B)) +(Card (A left join B on A.field = B.field) - Card (A inner join B on A.field = B.field)) * (Row_Size (A) +Size ("NULL") * Col_Num (B))*
Left out join (A right outer join B on A.*field* = B.*field*)	*Card (A inner join B on A.field = B.field)*(Row_Size(A) +Row_Size (B)) +(Card (A right join B on A.field = B.field) - Card (A inner join B on A.field = B.field)) * (Row_Size (B) +Size ("NULL") * Col_Num (A))*
Full out join (A full outer join B on A.*field* = B.*field*)	*Card (A inner join B on A.field = B.field)*(Row_Size(A) +Row_Size (B)) +(Card (A right join B on A.field = B.field) - Card (A inner join B on A.field = B.field)) * (Row_Size (B) +Size ("NULL") * Col_Num (A)) + (Card (A left join B on A.field = B.field) - Card (A inner join B on A.field = B.field)) * (Row_Size (A) +Size ("NULL") * Col_Num (B))*
Cross join (A cross join B)	*(Card (A) *Row_Size (A)) * (Card(B)* Row_Size (B))*

Table 2 – Join Query Quantity and Type

In the table 2, Row_Size () is the quantity of each row, Size ("NULL") is the length of string of NULL, and Col_Num () is the number of column in each row.

3.1.3 Estimate Total Response Time

After the algorithm estimates the quantity of transmission data, it estimates the total response time for each possible execution plan. In the join query execution process, the data transmission and local join query execution have the most impact on the total response time. Thus, the total response time is equal to the sum of time spent on data transmissions and query executions. Table 3 shows the total response time for those three possible execution plans.

Execution Plan	Total Response Time
Plan 1	$T_{(a, b)}(d) + T_{(b, c)}(d) + P_c(d)$
Plan 2	$T_{(b, a)}(d) + T_{(a, c)}(d) + P_c(d)$
Plan 3	$MAX(T_{(a, c)}(d), T_{(b, c)}(d)) + P_c(d)$

Table 3 –Total Response Time for Possible Execution Plan

In the table 3, $T_{(a, b)}(d)$ means the time spent on data transmission from machine A to machine B and $P_c(d)$ is the time for join execution on machine C. The time spent on data transmission in plan 3 is the maximum time of two transmissions because machine A and B send data to machine C simultaneously.

The time spent on data transmission between two sites is calculated as $\mathbf{T(d) = T_0 + d / S}$, where$T_0$ is the startup time, d is the number of bytes of transmission data, and S is the transmission speed between site A and B. The time for join execution on a machine is obtained from this function, $\mathbf{P(d) = P_0 + d / P}$, where, P_0 is the startup time, d is the number of bytes of data, and P is the process speed of the machine.

3.1.4 Network Transmission Speed and Server Process Speed

The server process speed and network speed that are used to calculate the response time in section 3.1.3 are computed with the statistics of join query execution history. The system records the quantity of transmission data and the time spent on data transmission and data execution for each join execution during the training period and the real join query executions. Then, the time for initialization, network speed between the source site and destination site, and processing speed are approximate by using the Least Squares Regression to fit on the history of measured times and saved for future time calculation.

3.1.5 Choosing the best execution plan

The algorithm compares the estimated response times of the three execution plans, chooses the one with the minimum estimated execution time, and then executes the plan using the three select queries mentioned in section 3.1.2

3.2 System Platform

My proposed service-oriented architecture is based on specialized grid services platform and software. Figure 9 shows the system platforms on which my system bases.

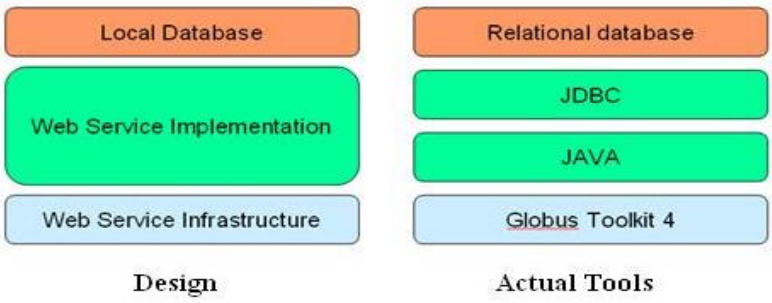


Figure 9 - System Platform

My web services are implemented using the middleware infrastructure provided by the Globus Toolkit 4.0. The open source Globus Toolkit [11] is a fundamental

enabling technology for the “Grid”, letting people shares computing power, databases, and other tools securely online across corporate, institutional, and geographic boundaries without sacrificing local autonomy. The toolkit includes software services and libraries for resource monitoring, discovery, and management, plus security and file management. These software services and libraries free developers from the underlying infrastructure development and increase the efficiency of system development.

All my web services are developed in the JAVA programming language. In the JAVA platform, there is a JDBC (Java Database Connectivity) [12] API which provides programmatic access to the data within a relational database. The advantage of using JAVA and JDBC for distributed database system is that they resolve the issues with heterogeneous platforms. Java programs can be run similarly on any supported hardware/operating-system platform. Each machine in my distributed database system manages its data with a Relational Database Management Systems such as MS Access, MS SQL server and MySQL.

3.3 System Architecture

My proposed join query optimization system comprises of five basic components: *join query recognizer*, *join query optimizer*, *joins query executor*, *speed calculator*, and *metadata database*. Figure 10 is the component diagram of the overall system.

3.3.1 Join Query Recognizer

The *Join query recognizer* checks if the user input query is a valid join query. This recognizer firstly detects if the input query has any syntax errors and if the columns, tables and database in user query exist in the distributed database. If the recognizer detects any error in user input query, recognizer will terminate the query execution and

returns an error message. The error detection prevents the network and computation resources from being executing an invalid query.

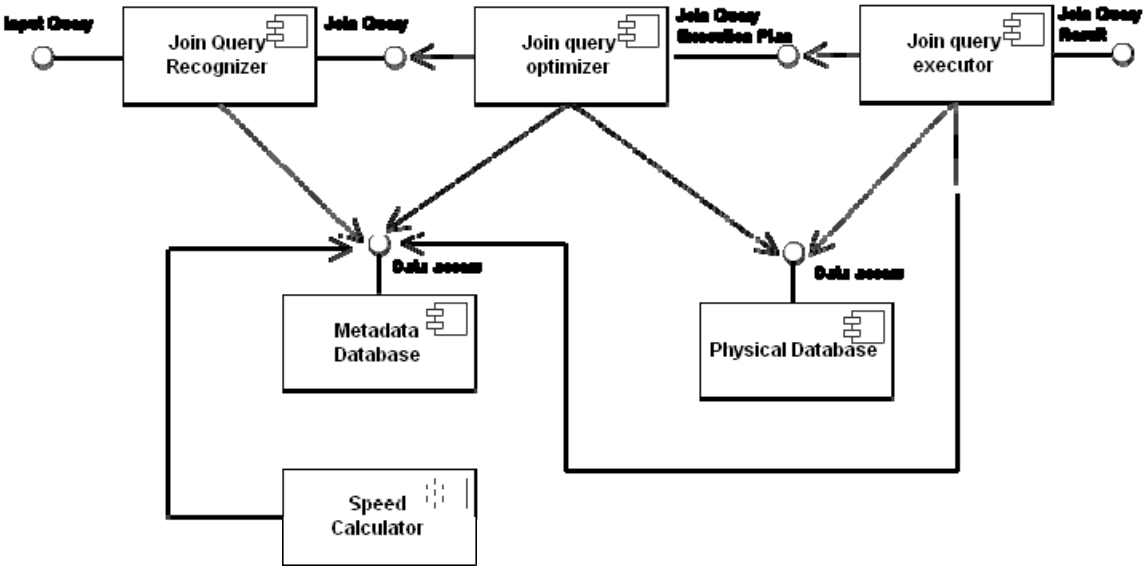


Figure 10 - System Component Diagram

The Join query recognizer also checks if the user query is a valid join query or not. If the query is not a valid join query, this query will not be executed using the join query optimization algorithm because it is more efficient to execute non-join query directly.

Furthermore, the join query recognizer prepares the information related to a join query after it detects input query is join query. First, it determines which type of join query the input query is. The type of join query is used to estimate the quantity of the result. Also the recognizer divides the query into three sub-queries as discussed in section 3.1.2. Finally, the recognizer retrieves the IP addresses of the machines, the database names and the types of database platforms where the source tables reside. The IP addresses and database names help the system locate the web services and data. The system can choose the correct JDBC driver for the corresponding type of database platform. The activities diagram of the join query recognizer is shown in appendix D.

3.3.2 Join Query Optimizer

The *Join query optimizer* is the core component in the whole system. The optimizer implements my proposed join query optimization algorithm. There are two sub-components in the optimizer: Local optimizer and global optimizer. The interaction between local optimizer and global optimizer is shown in figure 11.

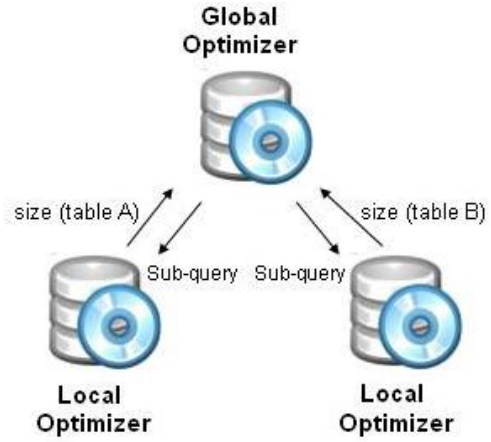


Figure 11 - Local Optimizer and Global Optimizer

Local optimizer collects the information on two join query participating tables. The local optimizer receives the sub-query from global optimizer, and executes the sub-query in its local database management system to determine the number of rows, number of columns, and the total quantity of columns. Also, local optimizer performs the query which requires the distribution information of sub-query (table 1), and sends the distribution information to global optimizer.

The global Optimizer controls the entire of join query optimization and chooses the best execution plan. The global optimizer interacts with the local optimizers which reside on the remote sites to retrieve the information from the two sub-query result. Then, the global optimizer estimates the size of the join query result based on the type of join query and the distribution information of the sub-query, and calculates the estimate of the

total response time for the three possible execution plans based upon the network speed, process speed and the time for initialization from metadata database. Finally, the global optimizer selects the plan that has the minimum response time as the best execution plan.

3.3.3 Join Query Executor

The *join query executor* performs the join query using the execution plan chosen by the join query optimizer and records the time spent on data transmission and local query execution. There are three sub-components in join query executor, the data sender, the data receiver and the global query executor.

The data sender collects the data for creating a temporary table from sources table. The data sender executes the sub-query and save the information includes the number of rows, the number of columns, the column names, and the column data types. The data receiver retrieves this data and creates a temporary table on the local database system.

Next, the data sender executes the sub-query again and transfers the resulting data to the data receiver. The data receiver receives the data and inserts it into the temporary table.

The data transmission between the data sender and the data receiver is divided into several smaller transmissions when the amount of resulting data is too large. If the size of object that holds the data exceeds the capacity of the machine's memory, the system will respond very slowly or crash. Thus, the system must divide the query into a set of sub-queries such that the quantity of data from the sub-query fits to memory size. Furthermore, it is more efficiency to run a set of small queries than one large query. Even if the large size of the query result does not exceed the memory capacity, it still consumes

a significant amount of memory. If the machine's free memory gets too low, the system will perform very slowly.

The global query executor interacts with the data sender and data receiver to perform the execution plan. The global query executor calls the data senders on the remote sites to prepare the result, and then instructs the data receiver on the destination site to retrieve the data and store it. After the data transmission completes, the global query executor sends the join query to the data sender on the site where both participating tables now reside. The data sender executes the join query and returns the result to global query executor.

In addition, global query executor records the response time for data transmission join query execution, and the quantity of data. For each transmission, the system records the source site, destination site, quantity of transmission data, and response time. The data are saved into the metadata and used to update the transmission speed and processing speed.

3.3.4 Speed Calculator

Speed calculator calculates the data transmission speed between sites, the processing speed of each machine, and the initial time for data transmission and query execution. The speed calculator retrieves the history execution data from metadata database, applies the *Least Squares Regression Algorithm* to fit the data, and then inserts new records or updates the existing records in the metadata database.

3.3.5 Metadata Database

A *metadata database* is a centralized database that stores the metadata of the entire distributed database. The metadata database stores the information of all database systems within the distributed database, such as the IP address of each database, the

database platforms, the database names, the database file directories, the table names, the column names, the data types, and so on. The database metadata helps the system to locate data, access data, and test for errors. It enhances the efficiency and effectiveness of database collaboration.

Moreover, the metadata database records the transmission speed, processing speed and history performance data. The global optimizer on each site uses this data to estimate the response time and choose an execution plan. Figure 12 is a relationship diagram of a simple metadata database.

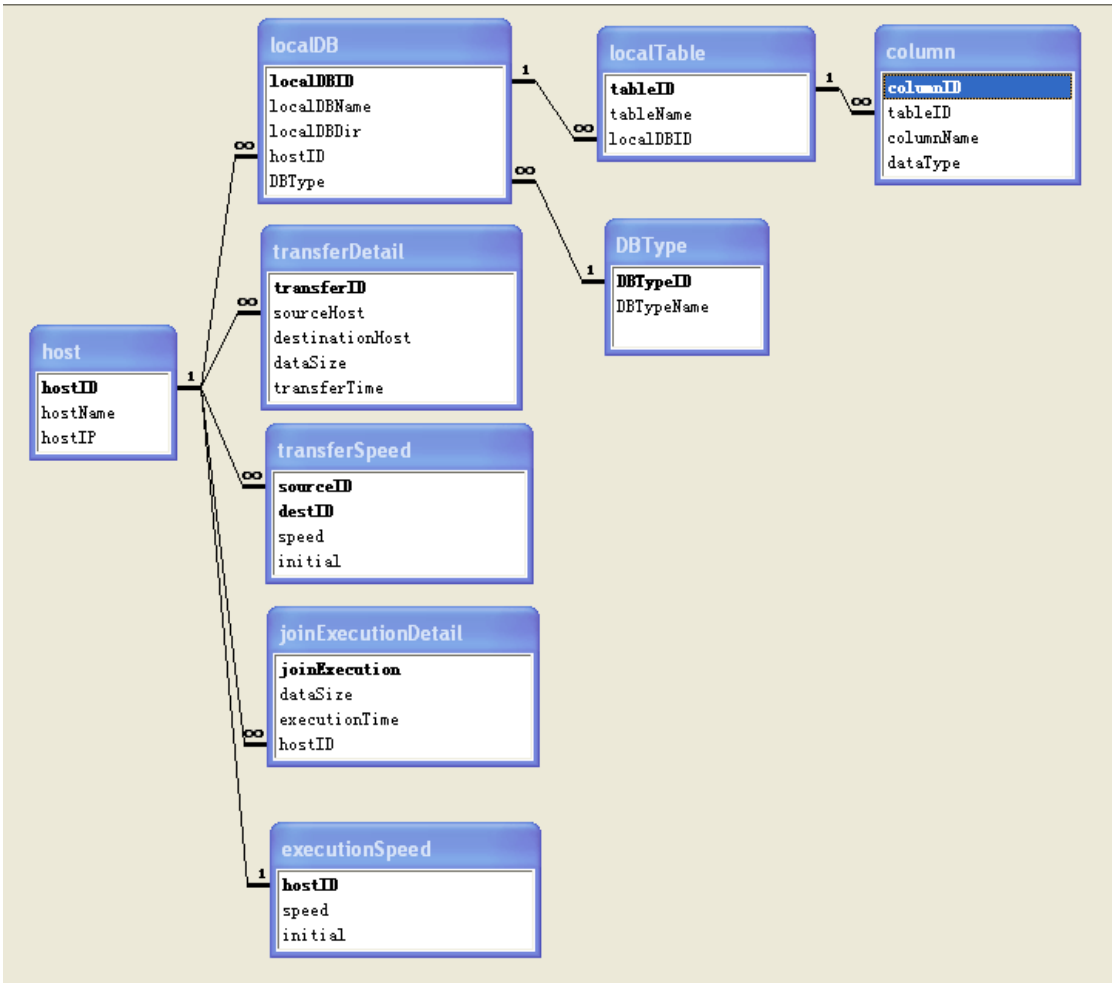


Figure 12 - Relationship Diagram of Metadata Database

Chapter 4: System Development

In this chapter, I will present the characteristics of the Unified Process model. In addition, I will show how my system development is benefited from the Unified Process model. Finally, I will describe my system development activities in the four static phases of the Unified Process model.

4.1 Unified Process Model

The *Unified Process* is a software engineering process model which is a simplified description of the set of software development activities and associated results. The Unified Process presents the dynamic and static views of the software process and creates and maintains models in the Unified Modeling Language (UML).

The Unified Process provides a static phase model which identifies four discrete phases, inception, elaboration, construction and transition. Moreover, the Unified Process provides the dynamic process activities in each phase. The Unified Process is unlike the waterfall model in which process activities occur in sequence. Software specification, design and implementation, validation and evolution are performed iteratively and can be developed in each phases. Figure 13 shows the Unified Process systems development lifecycle.

The Unified Process provides a graphical representation of the system using the UML. The UML is an object-oriented modeling language that allows the developer to clearly show the system requirements, architectures, and designs. The UML is now maintained by the standards organization Object Management Group (OMG). [13]

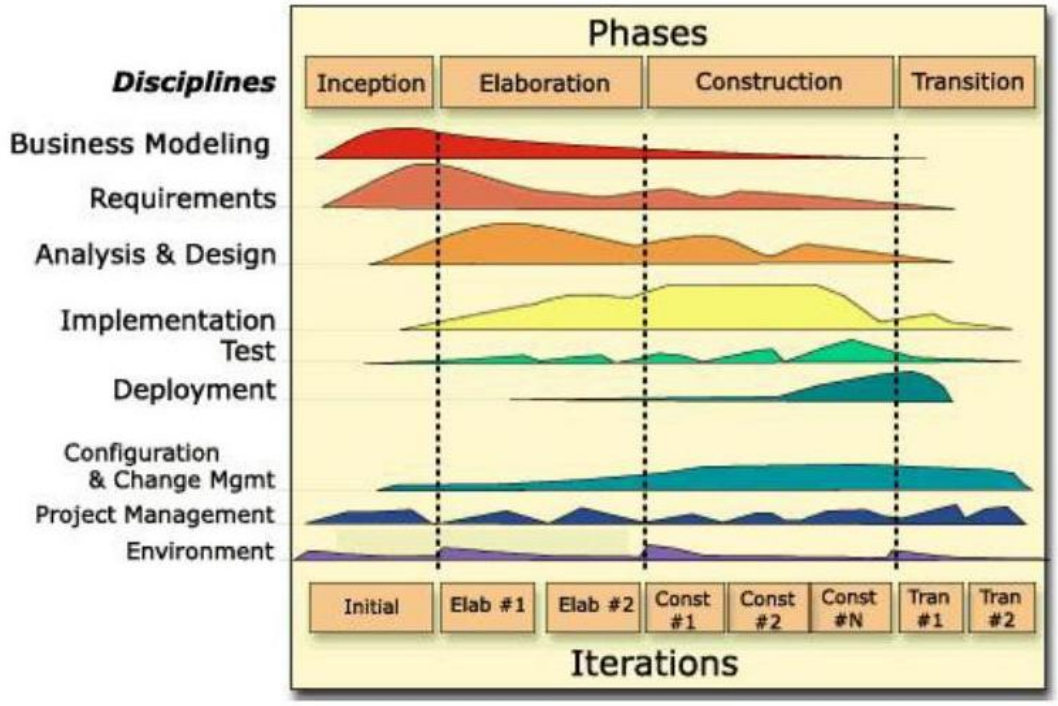


Figure 13 - Unified Process systems development lifecycle

4.2 Benefits of Using Unified Process

My project is developed by following the Unified Process. The Unified Process has three major benefits to my project development process as below:

1) *Improved planning.* I can easily establish a clear and reasonable project plan by following the Unified Process's static phase model. In the Unified Process, each phase is concluded with a well-defined milestone—a point in time at which certain critical goals must be achieved. [14] This static phase model helps me to setup the timeline, allocate the resources, and prioritize tasks for the entire project before I implement my project.

2) *Iterative software development.* First, iterative development can reduce project risk. In the Unified Process, the core and high risk items are addressed at the beginning of each phase. Those items are developed and tested firstly, so the risk can be mitigated before it affects other items. Also, it is easier for a developer to discover problems in

small sections of system rather than in a complex system. The quick troubleshooting can increase the speed of development. Second, iteration helps me improve the quality of system. It is difficult for a developer to sequentially perform in one run all the activities of locating all the potential problems, designing an error free solution, building the system and then testing at the end. During system development, I have an increased understanding of the problem and discover new factors which impact the join query performance. Therefore, I can improve the system performance by revising my system components based on the new findings, which I did not understand during the stage of system design.

3) *Visual model representation.* My system architecture, components and system activities are all represented in UML. The graphical representation of UML is more clear and understandable than a detailed natural language description. My system includes several interactive web services and a set of information which are transferred between remote sites. It is very important to depict the component interactions and information flow in graph, so I can make sure the interaction of web services is designed correctly and can determine if my code is consistent with my design. In addition, the UML diagrams help others understand my system and avoid the ambiguous communication. Examples of a UML diagram for my system are figures 10, figure 14 and appendix D.

4.3 Description of Software Development Activities

4.3.1 Inception Phase

During inception phase, one main goal is to identify the needs of the system and define the scope of the project. First, I studied the benefits of using distributed databases in business and the problem of join query performance on the distributed databases. From this study, I was able to establish the goal of my project which is to find out the best

execution plan for the join query that retrieves data from two different machines. Then, I researched the work related to join query optimization and understood the advantages and disadvantages of current methods. Finally, I developed an initial solution to achieve my project goal.

Another goal of the inception phase is to identify the resources needed for my project. One of the project risks is the lack of resources. The project could fail due to inadequate or unavailable resources. Human resource is very important to my project, so I designed my committee at first. A group of professors who are experts in the areas of distributed computing, database systems, system design and analysis, and software engineer can provide me the knowledge and experience in those areas and help me finish the project. Also, I sought the availability of development software and machines. In my project, a set of web services will run on different machines which are connected via the network. A tool for web services development and management as well as an environment in which to test these services are critical to my project. The Globus toolkit provides the tool for web services development and management. I was able to use services on the computers in New Hanover Community Health Center as well as my own computer.

4.3.2 Elaboration Phase

The goals of the elaboration phase are to design a system architectural framework, develop the project plan and identify key project risks.

First, I decided my system is based on the service-oriented architecture. My system is divided into several interactive web services. The functions of web services are described in English. A component diagram shown in figure 10 is developed to show the structure of system and the connections between web services. Also, I use a sequential

diagram (Figure 14) to capture the methods in web services, the collaboration of these methods, and the messages that are passed between web services.

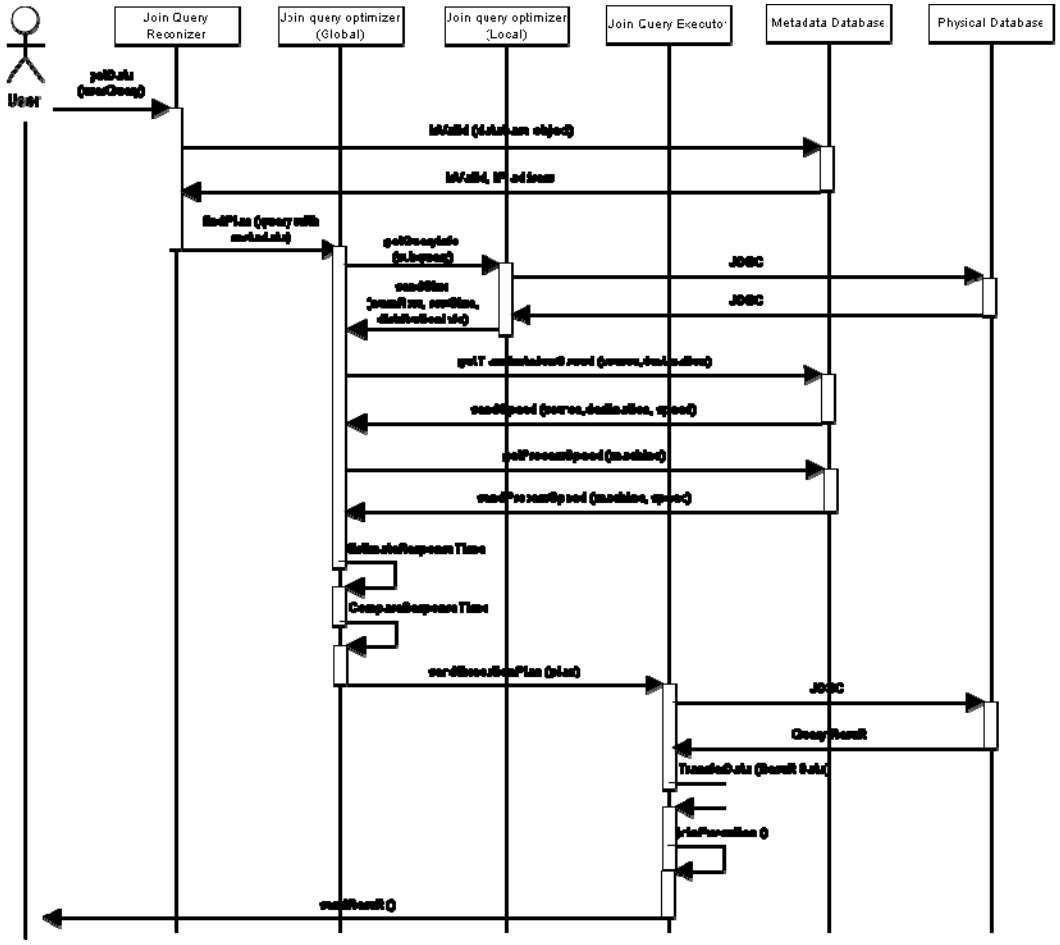


Figure 14 - Sequential Diagram

The next step in elaboration phase is to establish the project plan. The functions to be developed are prioritized by their risks to the system. The most risky function will be developed and tested first. A project timeline is established based on the function priority order. At the end of the elaboration phase, project committees reviewed my problem scope, project solution, and project plan.

4.3.3 Construction Phase

Construction is the main phase for system development in the Unified Process. The main tasks of construction are to design, develop and test the system. A working

system should be ready after this phase. In my project, there are three iterations for system development.

Iteration 1 - The goal of this iteration is to eliminate the risk of system platform compatibility. My system includes several different system platforms which are the Globus toolkit, JDBC and MS SQL server. It is very important to understand how to make these systems work together. I developed a simple web service which can manipulate the data stored on a MS SQL server.

Iteration 2 - In the iteration 2, I developed the Java programs for my system. Each service is developed as one class. All the methods are developed in order of their priorities. In this iteration, I confirmed the system can perform the join query which retrieves data from two remote sites, and then I implemented the functions of optimizing the join query execution. Here is my system development order:

1. Transfer data between databases
2. Execute join query
3. Retrieve the quantity of resource table
4. Estimate the quantity of result size
5. Measure time to transfer data
6. Calculate the speeds using the Least Squares Algorithm
7. Design Metadata database

Initially, I developed a prototype that was not actual web services. The purpose was to test and debug the algorithms before developing them in a more complicated system required for web services.

Iteration 3 – The task of this iteration is to transit my Java program to the web service platform. All the web services description files are completed for the corresponding Java programs and all the Java objects are changed to be web services.

The implementation of parallel processing of the join query was also developed in this iteration. In iteration 2, I already completed the function of data transmission between sites. The implementation of the parallel process is such that the web services of data transmission run in two threads to retrieve data from two machines simultaneously. A key requirement of this implementation is that the web services must be able to store the two sets of data from source machines at the same time. If this does not happen, then either the transfer will not be in parallel or data will be lost. The Globus toolkit allows one to create multiple resources for one web service so the data of each transmission can be stored separately.

At the end of the iteration 3, all of the Java programs are moved to the Globus platform and a working system based on web services is established.

4.3.4 Transition Phase

Transition is the final phase of Unified Process in which the system is moved from development environment to the real business environment. In this phase, the developer ensures that the system runs correctly and provides users training on the system.

In this phase, I setup and tested all the web services on the test machines. Because the test machines use the same type of operating system and database management system, no code was modified when all the web services were moved from development environment to the test environment.

After I finished the system installation on test machines, I ran three main tests on my system which evaluates the performances on data transmission, parallel data transmission and my join query optimization algorithm. During the system test, several factors which impact system performance were found. A set of system development activities were implemented to improve the system performance in this phase. The details of these tests will be described in chapter 5.

Chapter 5: System Test

5.1 System Test Environment

The goal of my project is to attempt to determine the best query execution plan for the query which retrieves data from two different sites, so my system test is run on three machines, which reside at two different locations. Table 4 shows the hardware and software configuration of these three test machines.

Machine No.	Type	CPU	Core #	Memory	OS	DBMS
1	Server	Intel (R) Xeon (R) CPU L5410 @ 2.33G Hz	Quad	4G	Window Server 2003	SQL Server 2000
2	PC	Pentium (R) CPU E5200	Dual	3G	Windows XP Professional	SQL Server 2008 Express
3	PC	Genuine Intel (R) CPU T1300 @ 1.66GHz	Single	2G	Windows XP Professional SP3	SQL Server 2008 Express R2

Table 4 – Machine Configuration

Machines 1 and 2 are located at site 1 which is the New Hanover Communicate Health Center at 925 North Fourth Street Wilmington, North Carolina. Machines 1 and 2 are connected to the local area network in the center. Machine 3 is my personal laptop is at site 2, which is close to the University of North Carolina Wilmington. Site 1 communicates with site 2 using the dial-in VPN through the Internet.

Microsoft SQL Server is installed on each test machine. There are two tables in each database, employee and department tables which have two common and related fields, department ID and branch ID. The department table has 150 records, and the employee table has 500,000 records. The names of each employee and department were generated randomly.

5.2 Test 1: Data Transmission between Machines

Test 1 is to test the data transmissions between machines. The data transmission is the fundamental function of my system. The process of data transmission includes reading data from the database system on the source machine, transferring data from the source site to the destination site, and inserting data into the database system on the destination machine.

In test 1, I select data from the source machine in the range 1 to 500,001 rows in increments of 50,000 rows, transfer the data to the destination machine, and record the time spent on each step.

5.2.1 Total Response Time

Figure 15 shows the total response time of data transmission from machine 2 to machine 3. The total response time includes the time for querying, transferring, and inserting data. We can see that the total response time increases as the number of rows transferred increases. In addition, the dominant components in the total response time are the time for transferring data as well as the time for inserting data. Therefore, join optimization algorithm should consider the cost on network transmission and the speed of the machine.

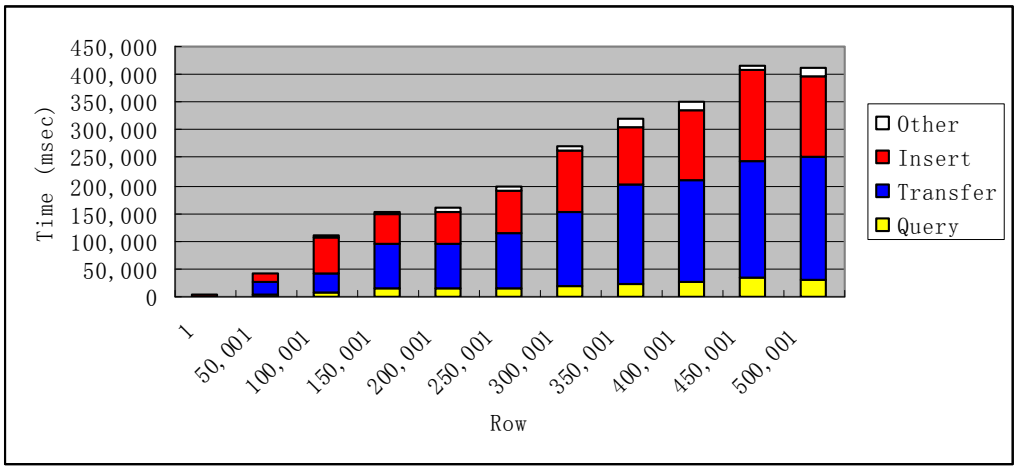


Figure 15 - Total Response Time of Data Transmission from Machine 3 to 1

5.2.2 Time for Querying Data

At the beginning of data transmission, the system loads data from the source database into a string or byte stream. In figure 15, we can see the time for reading data has a small impact on the total response time.

However, it is very important to control quantity of data read from the database with each query. The reason is that a query, which results in such a large amount of data that it exceeds the memory limits of the machine, results in excessive swapping or worse a system crash. Therefore, the quantity of transmission data must be less than the size of memory on the source and destination machine.

Moreover, controlling the quantity of data selected from MS Access can increase the performance. MS Access has a low performance on selecting large amounts of data. I ran a small experiment to tests how the quantity of data affects the performance on selecting data. I selected 100,000 rows of records from a MS Access database. It takes 633 seconds to load the data into string buffer when I selected all the rows in one query. The execution time decreases to 97 seconds when I set the memory limitation size as 10,000 bytes and performed 725 queries selecting 138 rows each. The performance

increased 650%. This performance issue only happens on MS Access and not with MS SQL server.

The solution to controlling the size of data stream is to divide the query into a set of sub-queries which contains a smaller range of result records. The systems calculates how many rows of records can be stored in a reasonable amount of memory as following, Number of Rows = Floor (Memory Size / Row Size). After that, we can select that number of rows each time by using the *between* operator in the *where* cause. For example, the query, "SELECT * FROM employee WHERE rowID between 1 and 100", returns 100 rows of records. As a result, the size of the result can stay within the size of memory. This method requires each row has a unique number so that system can identify the row number.

Figure 16 shows the time querying 100,000 rows of records from a MS Access database on the machine 3 with different sizes of sub-queries. When the sizes of sub-queries are small, the time for querying data is long because the system connects and accesses the database many times. When the sizes of sub-queries are large, the string buffer object and the JDBC result object occupy a great amount of memory, so the time for querying data is also large. In other words, 100,000 rows can be transmitted as a single message with 7,200,000 bytes or 720 messages of 10,000 bytes each. The question is what the right size to use is. We can see the response time for querying data is not the best when the sub-query size is 10,000 bytes. The response time for querying data can be decrease to 24 seconds when the sub-query size is 90,000 bytes.

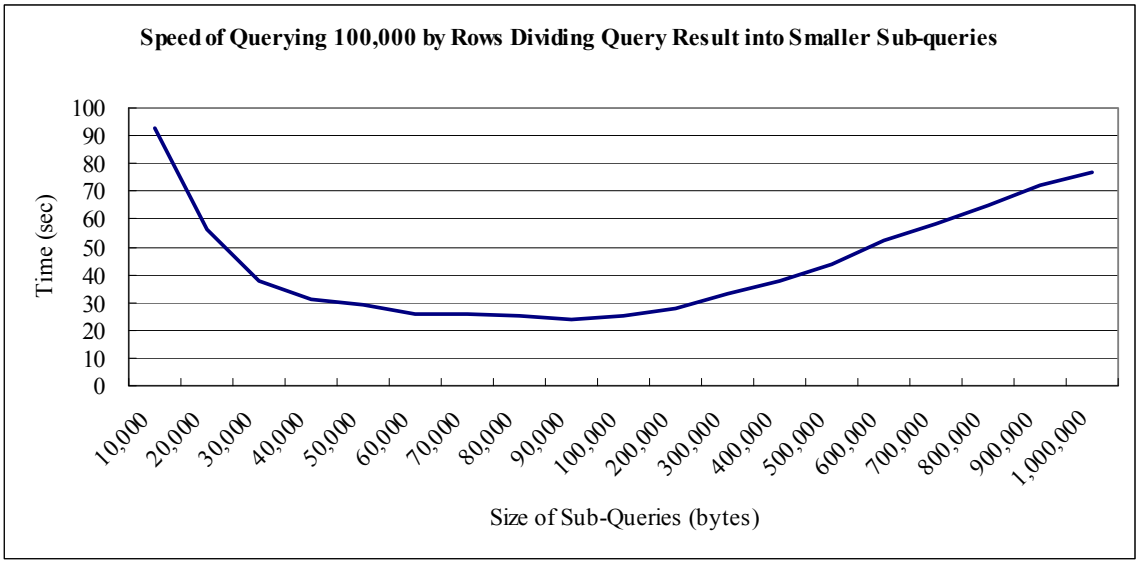


Figure 16 - Query Time and Memory Size

5.2.3 Time for Transferring Data

In the figure 15, we can see the time for transferring data is one of the major costs in the data transmission process from machine 2 to 3. This time increases as the quantity of transmission data increases.

This time is also determined by the network speed between source machine and destination machine. In the figure 17, the time for transferring data from machine 2 to 1 is much shorter and more stable than from machine 3 to 1. That is because machine 2 and 1 communicate in a 10/100M byte local area network and machine 3 and 1 transfer data to each other using dial-in VPN through the wide area network.

Although the time for transferring from machine 3 to 1varies because the wide area network condition is not stable, we still can see the time for transferring data is linearly increasing as the size of transmission data increases. Therefore, it is reasonable to calculate the time for transferring data as Time = T_0 + Data Size / Network Speed, where T_0 is the startup time.

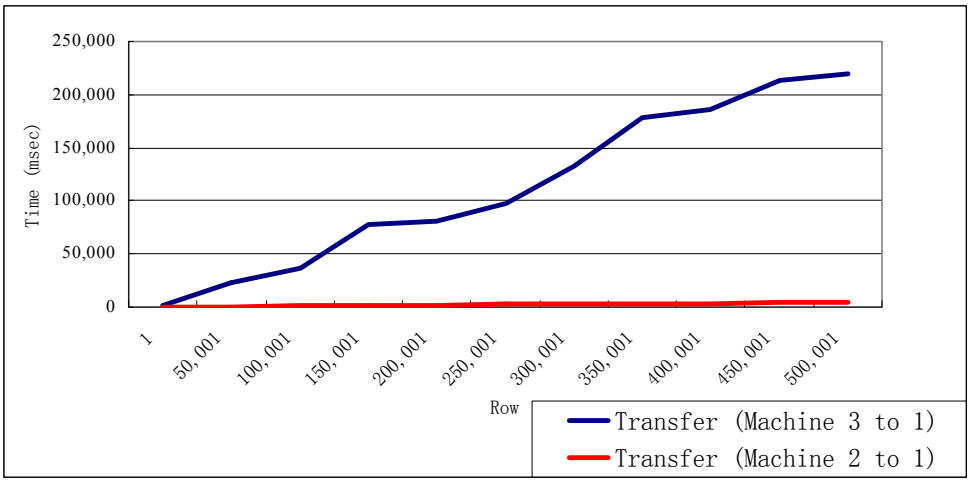


Figure 17 - Transmission Time from Machine 2 to 1 and from Machine 3 to 1

5.2.4 Time for Inserting Data

After the data is transferred from source machine to destination machine, the system inserts all the data into the destination database. The speed of inserting data on each machines are different. In average, machine 1 can insert 333 rows per second, machine 2 can insert 488 rows per second, and machine 3 can insert 185 rows per second. The difference of inserting speed impacts the time for inserting data, so system cannot ignore the influence of inserting data.

Because the network between machines 1 and 2 is a 10/100M bytes high speed local area network, the time spent on transferring data is very short compared to the time for inserting data as well as the transfer between machines 3 and 1 as shown in figure 15. The time for inserting data is the most significant part of total response time when these two machines are involved. The time for inserting data takes 56% of the total response time for the data transmission between machine 2 and 3. Figure 18 shows the total response time of the data transmission from machine 2 to machine 1. We can see most of

time is used to insert data in each data transmission. The time for inserting data takes 91% of the total response time for the data transmission from machine 2 to 1.

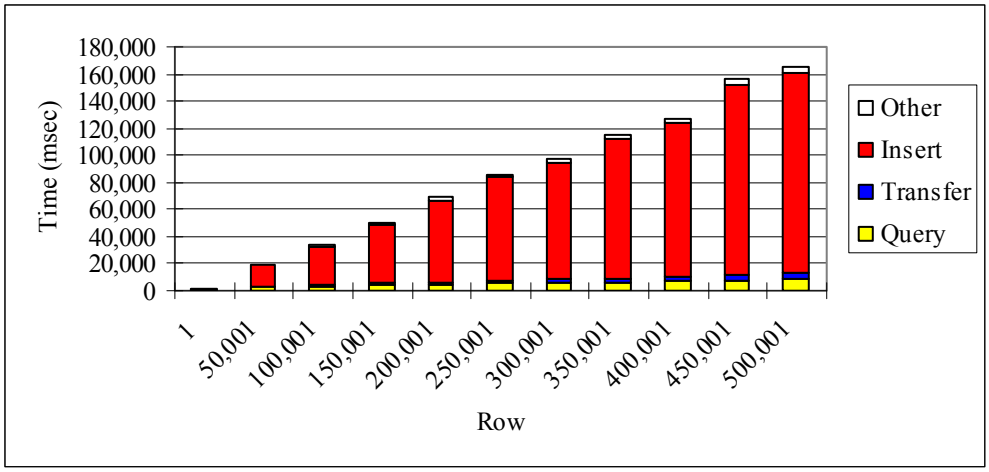


Figure 18 - Total Response Time on Data Transmission between from 2 to 1

In order to optimize inserting data, I changed the method of inserting data. In the beginning, I inserted data into database row by row. However, row-by-row insertion is very inefficiency. I found that batch insertion is faster than row by row insertion. I tested these two insert methods on machine 2 by inserting different numbers of rows into the local database and recording their execution time. Figure 19 shows the execution time of batch insertion and row-by-row insertion. We can see that batch insertion is always faster than the row-by-row insertion. In average, the batch insertion is 35% faster than the row-by-row insertion.

Although batch insertion can decrease the time of inserting data, the time for inserting data is still significant to the total response time. One of the reasons for the insert performance issue is because the database systems take time to build the index when it inserts data. However, researching how to optimize data insert performance is out of the scope of my project.

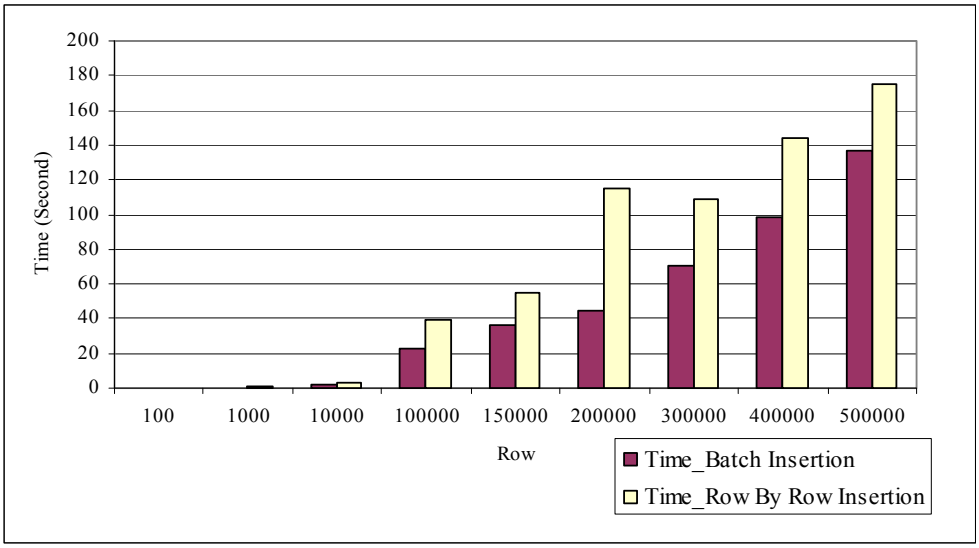


Figure 19 - Batch Insert vs. Row-by-row insert

5.3 Test 2: Total Response Time of Parallel Process

In the test 2, I measured the performance of the parallel query process. First, I performed the parallel query process on machine 1 which is a multi-core machine to retrieve the data from machines 2 and 3 simultaneously and recorded the total response time. Then, for comparison I executed two data transmissions in sequential to retrieve the same amount of data from machines 2 and 3 and recorded the total response time for these two data transmissions.

Figure 20 shows the response time for the parallel query process and two sequential data transmissions. We can see the time for parallel query process is approximately equal to the time for data transmission from machine 3 to 1 for which the transfer speed is slower than between machines 2 and 1. The reason why the time for parallel process is not the same as the time for the data transmission from machine 3 to 1 is because the network speed between machine 3 and 1 is not stable. Furthermore, the time for the parallel query process is less than the sum of time for those two sequential

data transmissions. That means that there is potential for the parallel query process to execute faster than sequential queries.

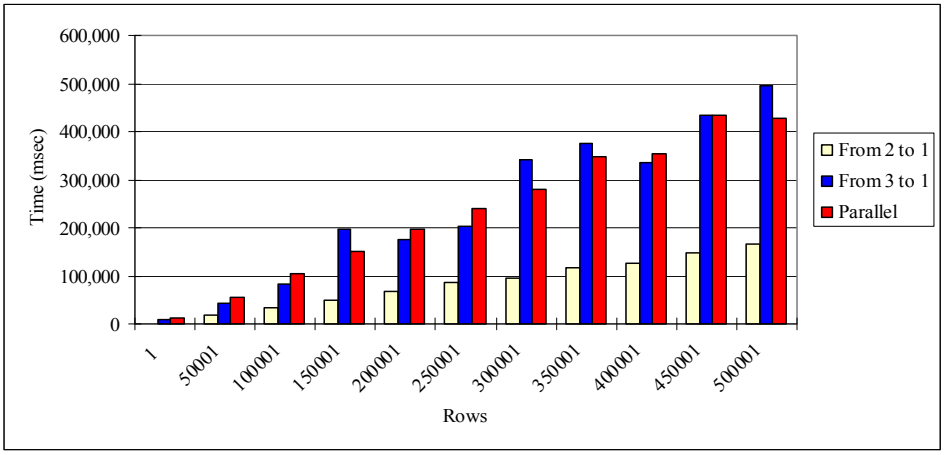


Figure 20 - Parallel Process on Multi-core machine

5.4 Test 3: Join Query Optimization

In test 3, I tested the join query performance of my join query optimization algorithm. In this test, I executed a join query on machine 1 to retrieve data from machine 2 and 3. The test join query is "SELECT * FROM department INNER JOIN employee ON department.departmentID = employee.departmentID". I selected the data from employee table on machine 2 between 1 row and 300,001 rows in increments of 50,000 rows, and selected 50 rows and 150 rows of data from department table on machine 3. The test join query has three possible execution plans shown in figure 21.

5.4.1 Performance of Three Execution Plans

After I ran the test, the response time of those three execution plans are recorded and shown in figures 22 and 23. The response time for plan 1 is the longest one in those three execution plans. Plan 2 has the best performance when joining different number of employee records with 50 records. Plan 3 executes the join query faster than plan 2 when joining different number of employee records with 150 department records.

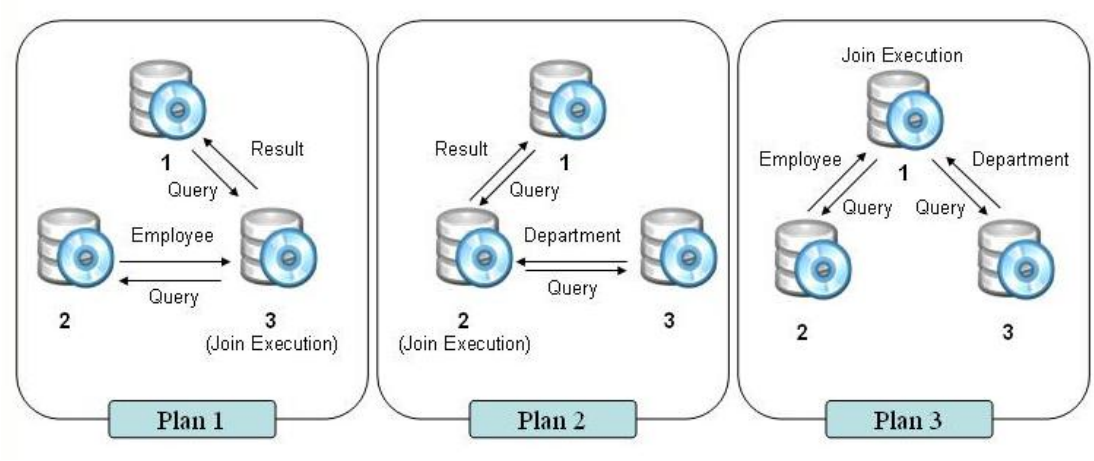


Figure 21 - Three Execution Plans for Test Join Query

Plan 1 is a sequential transmission which transfers the larger table of two join query participating tables and executes the join on the computer where the smaller table is stored. The response time of plan 1 is equal to $T_{employee} + T_{result}$, where $T_{employee}$ is the time for transferring the employee table and T_{result} is the time for transferring the data of result. In the figures 24 and 25, the sizes of the employee table and the result are large for the test join query. Thus, plan 1 is the worst plan in the three execution plans.

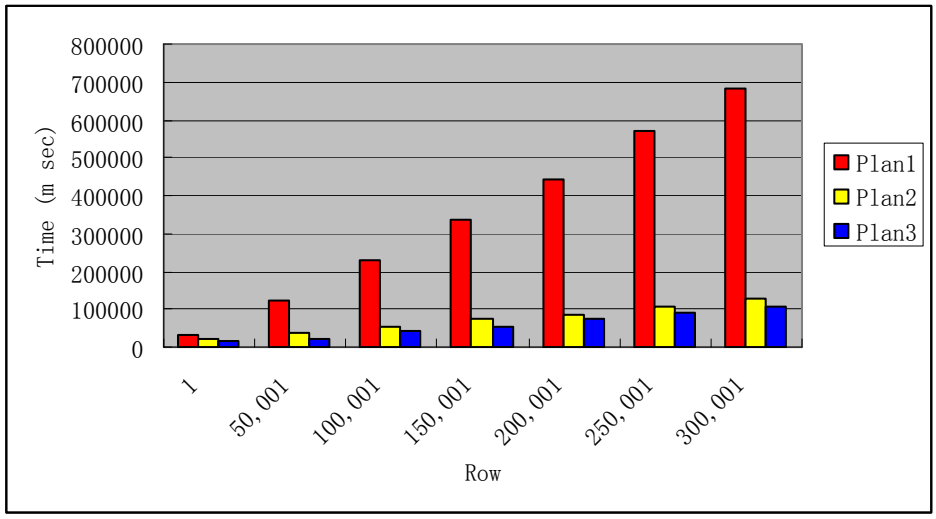


Figure 22 - Performance of three plans when selecting 150 department records

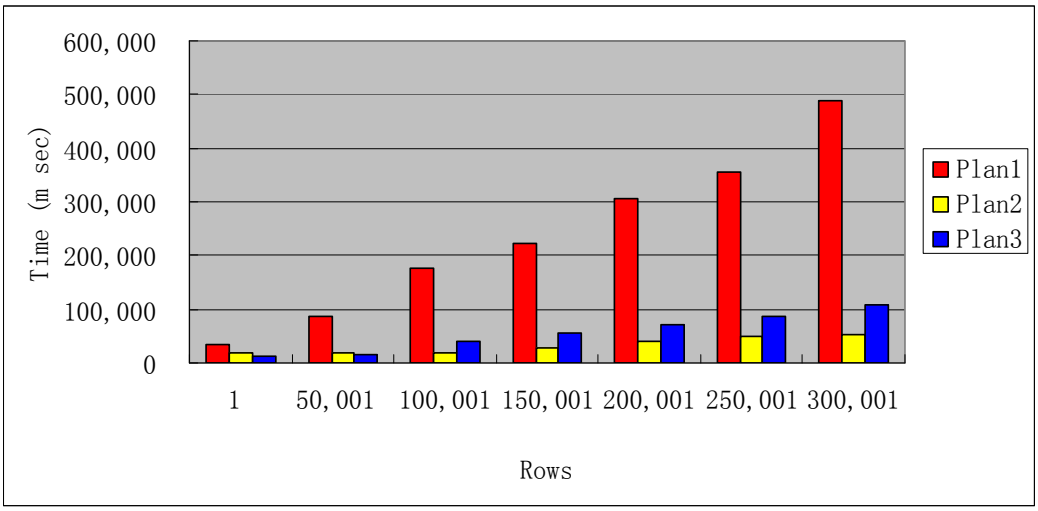


Figure 23 - Performance of three plans when selecting 50 department records

Plan 2 is a sequential transmission which transfers the smaller table of two join query participating tables and executes the join query on the computer where the other join query participating table resides. The total response time for plan 2 is approximately $T_{department} + T_{result,}$ where $T_{department}$ is the time for transferring department table and T_{result} is the time for transferring the result. From the result of this test, we can see that the data analysis which is discussed in section 2.3.1 can eliminate the worst sequential transmission plan by transferring the smaller table.

Plan 3 is the parallel query transmission. The response time for plan 3 is approximately MAX ($T_{employee,}$ $T_{department}$), where $T_{employee}$ is the time for transferring the employee table and $T_{department}$ is the time for transferring the department table. In this test, $T_{employee} > T_{department,}$ so the response time for plan 3 is $T_{employee.}$ When the join query selects 150 rows of department records, all the employee records which the query selects will be returned as a result because all the employees are in those departments. Figure 24 shows the size of the result is greater than the size of employee table, so $T_{result} + T$

$T_{department} > T_{employee}$. In this case, the join query which selects 150 department records is performed faster in plan 3 than plan 2.

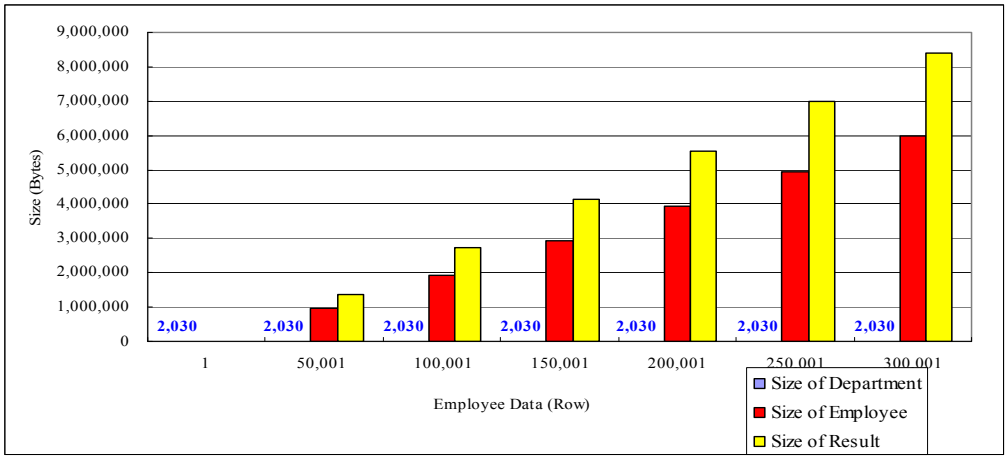


Figure 24 - Size of Department, Employee, and Result when Selecting 150 department records

When the query joins employee records with 50 rows of department records, the join query only returns the employees who are in those 50 departments. Figure 25 shows the size of the result is smaller than the size of the employee table, so $T_{result} < T_{employee}$. Also, the size of department table is much smaller than the size of employee table and result. Thus, the plan 2 is faster than plan 3 when the size of the result is much smaller than the maximum size of two join query participating tables.

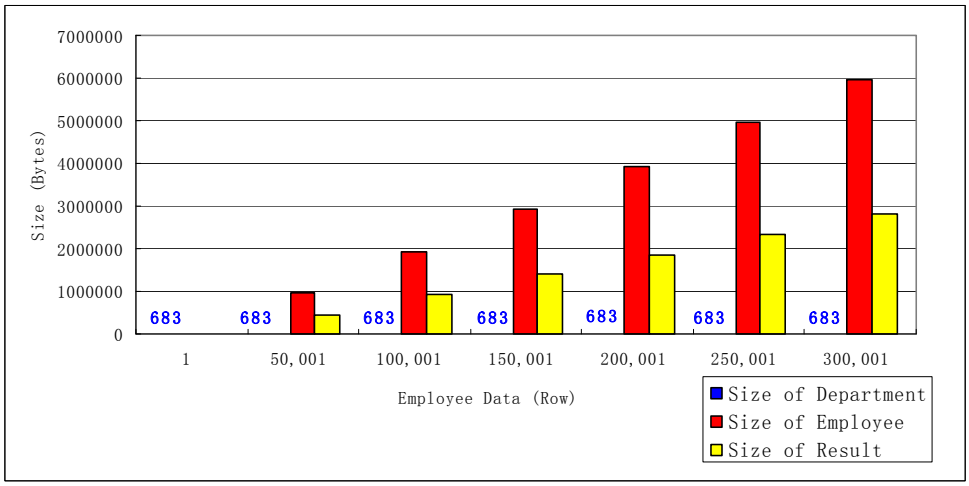


Figure 25 - Size of Department, Employee, and Result when Selecting 50 department records

However, we cannot find the best plan for all the join queries only by comparing the sizes of the transmission data for the different plans. Transferring more data can sometimes be faster than less data because of the effect of transmission speed. When the test query joins 50,001 employee records with 50 department records, the size of the employee table is 952,026 bytes and the size of the department table plus result is 458,310 bytes. Although the size of the employee table is larger than the size of result, plan 3 is better than plan 2. The reason is that the time for transferring department records from machine 3 to machine 2 is longer than the time saved by transferring a small result from machine 2 to machine 1. The transfer speed between machine 3 and 2 is so much slower than between machine 2 and machine 1. The sizes of the employee table and result are not so much larger than the size of the department table to overcome the slow transmission.

5.4.2 My Join Query Optimization Algorithm

First step of testing my algorithm is to derive a model to predict the time to query, transfer and insert using the Least Squares model. In test 1, which is described in section 5.2, I recorded the execution time for querying, transferring, and inserting and the quantity of transmission data for each training data transmission. Figure 26 shows the training result of transferring data from machine 1 to 2. All the training results are in Appendix A.

The results of speeds and startup time are shown in Appendix B. This information is stored in metadata database and used to estimate the total response time for each execution plan.

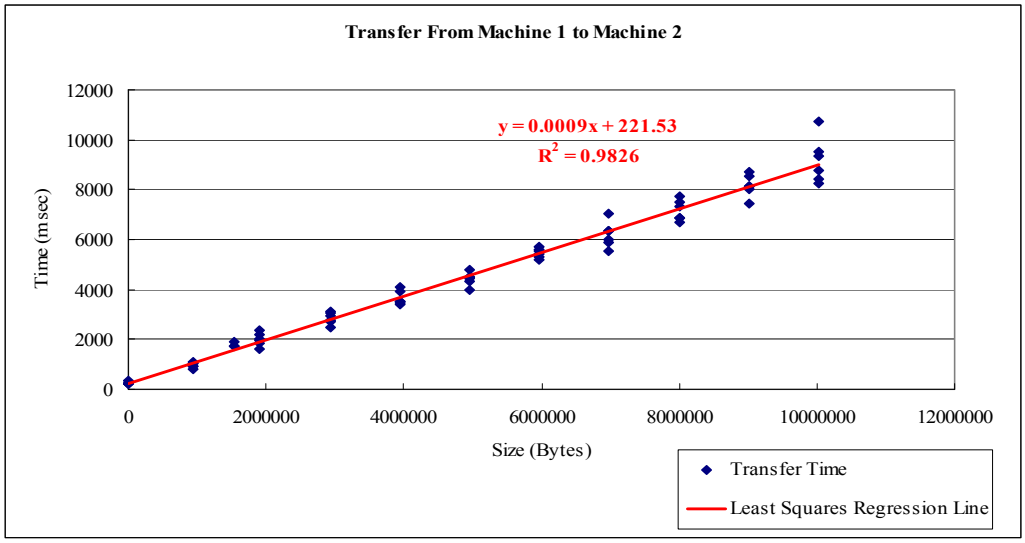


Figure 26 - Training Result of Transferring from Machine 1 to Machine 2

The second step is to test if my algorithm can estimate the size of transmission data. In this test, I executed the test join query which joins 150 department records with from 1 to 300,001 employee records in increments of 50,000 rows and recorded the size of actual transmission data. Then, I ran my optimization program to estimate the size of data and compared it with the size of actual data.

My algorithm can calculate the exact number of rows and estimate the approximate size of data. It is not efficient to sum the actual size of each column for a large number of records. The first method which I used to estimate the size of data is based on the maximum size of each column. The maximum size of column is the length which the database system allows the user to input in the column. It is very easy and quick to retrieve the maximum size of each column from JDBC. However, the size of data estimated by maximum size is much larger than the actual size, because most of the records are less than the maximum size. To improve the estimation result, I used the second method which calculates the size using the average size of the first 50,000 rows. In figure 27, we can see the estimated size of method 2 is much closer to the actual data

size than method 1. The reason I use the first 50,000 rows is that querying 50,000 rows takes about only 3 seconds which was measured in test 1. This cost time is short so that my program can estimate the size efficiently. The average percentage of difference between the actual size and the sizes estimated by method 1 and method 2 are 629% and 2.86% respectively. The estimated sizes for the other types of join queries are in the Appendix E.

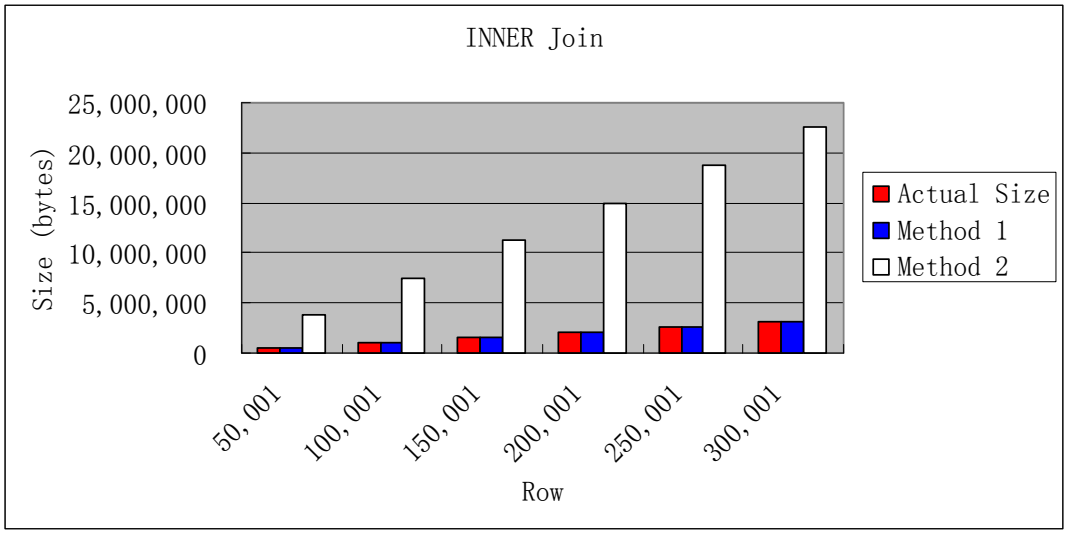


Figure 27 - Sizes of Join Query Result

After estimating the size of the transmission data, my optimization algorithm estimates the total response time for each plan, compares those total response times, and uses the plan with the minimum estimated time. Because the response time of plan 1 is the longest, I will just discuss the result of plans 2 and 3. The actual and estimated execution time for plans 3 and 2 are shown in tables 5, 6, 7, 8 and figures 28, 29, 30, 31. The actual time and estimated execution time for plan 1 is shown in Appendix C. The percentage of difference is (actual size – estimated size) / actual size. We can see the estimated time is close to but not equal to the actual execution time. There are three reasons cause the difference between estimated time and actual time.

One reason is that my algorithm does not include the startup time for the web services in the estimated time. The time cost of starting the web services is significant when the time of data transmission is small. This is the difference between the actual time and estimated time is larger as a percentage with smaller data size.

Another reason is the network speed varies from time to time. Although the Least Square can calculate the general speed between machines, the speed in a specific time is different with that general speed. In the table 5, the actual time is larger than the estimated time when selecting 150,001 rows from employee table. However, the actual time is less than estimated time when selecting 200,001 rows of employee records. Third reason is the error of estimated data size.

Department (# of Rows)	Employee (# of Rows)	Estimated Time (m sec)	Actual Time (m sec)	Difference (%)
150	1	-942	15,796	105.96%
150	50,001	24,350	37,106	34.38%
150	100,001	49,643	53,151	6.60%
150	150,001	74,936	79,180	5.36%
150	200,001	100,228	87,820	-14.13%
150	250,001	125,521	107,163	-17.13%
150	300,001	150,814	133,972	-12.57%

Table 5 - Actual Time and Estimated Time for Plan 2 when selects 150 department records

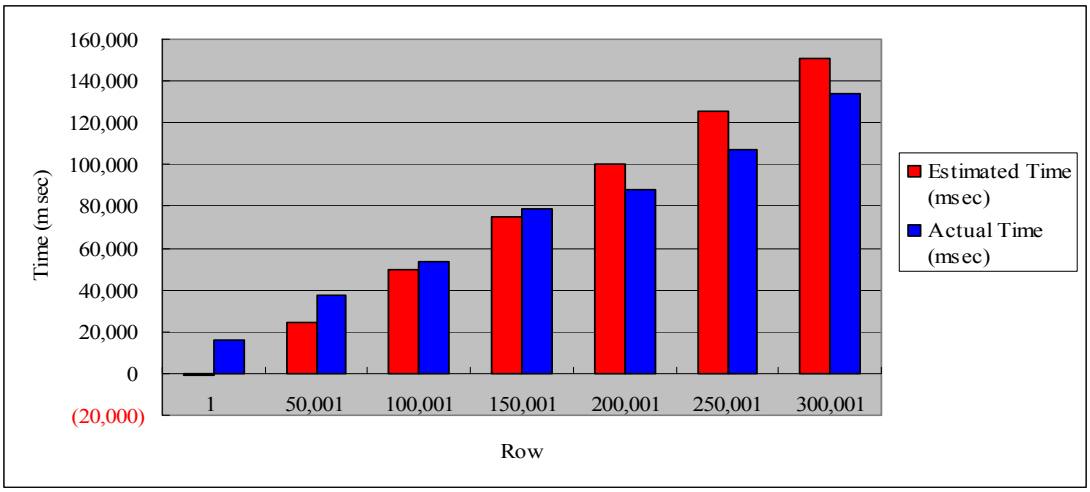


Figure 28 - Actual Time and Estimated Time for Plan 2 when selects 150 department records

Department (# of Rows)	Employee (# of Rows)	Estimated Time (m sec)	Actual Time (m sec)	Difference (%)
150	1	3,760	11,249	-66.57%
150	50,001	18,446	23,857	-22.68%
150	100,001	33,132	39,388	-15.88%
150	150,001	47,818	55,433	-13.74%
150	200,001	62,504	73,119	-14.52%
150	250,001	77,190	87,414	-11.70%
150	300,001	91,877	105,257	-12.71%

Table 6 – Actual Time and Estimated Time for Plan 3 when selects 150 department records

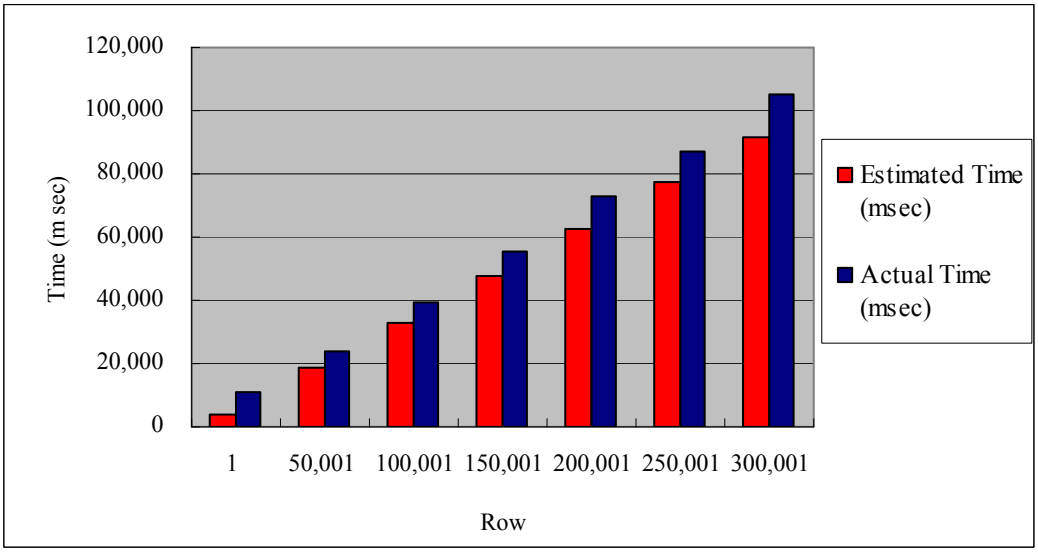


Figure 29 - Actual Time and Estimated Time for Plan 3 when selects 150 department records

Department (# of Rows)	Employee (# of Rows)	Estimated Time (m sec)	Actual Time (m sec)	Difference (%)
50	1	-997	18,952	105.26%
50	50,001	7,381	17,779	58.48%
50	100,001	15,762	19,481	19.09%
50	150,001	24,129	28,480	15.28%
50	200,001	32,433	38,776	16.36%
50	250,001	40,729	48,291	15.66%
50	300,001	48,994	52,181	6.11%

Table 7 – Actual Time and Estimated Time for Plan 2 when selects 50 department records

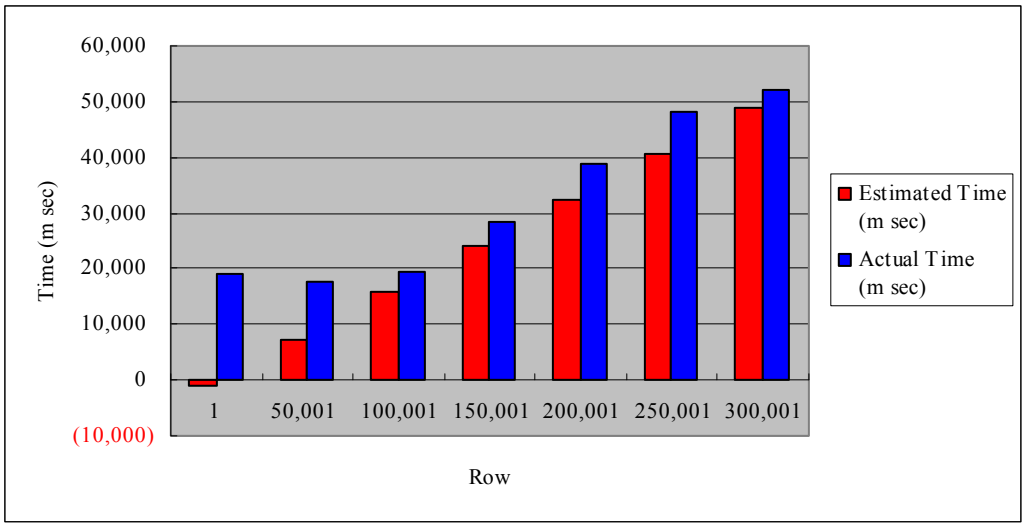


Figure 30 - Actual Time and Estimated Time for Plan 2 when selects 50 department records

Department (# of Rows)	Employee (# of Rows)	Estimated Time (m sec)	Actual Time (m sec)	Difference (%)
50	1	3,760	9,172	59.01%
50	50,001	18,446	24,811	25.65%
50	100,001	33,132	40,824	18.84%
50	150,001	47,818	56,167	14.86%
50	200,001	62,504	70,463	11.30%
50	250,001	77,190	85,383	9.60%
50	300,001	91,877	106,912	14.06%

Table 8 – Actual Time and Estimated Time for Plan 3 when selects 50 department records

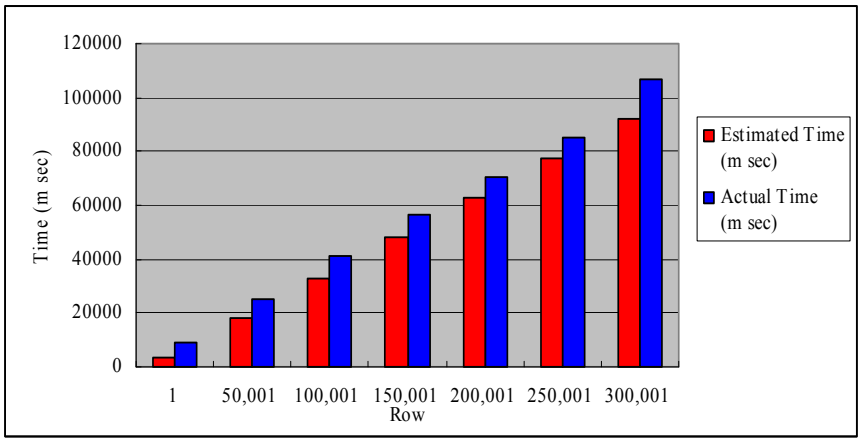


Figure 31- Actual Time and Estimated Time for Plan 2 when selects 50 department records

Although the estimated time is not exactly equal to the actual response time, my program can predict the best execution plan most of the time by comparing the estimated times of plans. The results are shown in table 9. The limitation of the system is that it

cannot predict the best plan for the join query which selects a small amount of records. However, the join query which selects a small amount of records is not time-consuming. This kind of join query can be executed in sequentially or parallel without using my algorithm.

Department (# of Rows)	Employee (# of Rows)	Plan Chosen	Best Plan	Correct?
150	1	2	3	**No**
150	50,001	3	3	Yes
150	100,001	3	3	Yes
150	150,001	3	3	Yes
150	200,001	3	3	Yes
150	250,001	3	3	Yes
150	300,001	3	3	Yes
50	1	2	3	**No**
50	50,001	2	3	**No**
50	100,001	2	2	Yes
50	150,001	2	2	Yes
50	200,001	2	2	Yes
50	250,001	2	2	Yes
50	300,001	2	2	Yes

Table 9 – The Result of Prediction

To analyze the efficiency of my join query optimization algorithm, I compared the execution time of my algorithm, plan 2, and plan 3. I do not compare to the execution time of plan 1, because the execution time of plan 1 is relatively large compared to my algorithm, plan 2, and plan 3. The execution time of my algorithm is equal to the time for predicting the best plan plus the time for executing the join query in the estimated plan.

In figures 32 and 33, we can see the execution time of my algorithm is similar to plan 2 when the join query selects 150 rows from the department table, and greater than plan 2 when the join query selects 50 rows from the department table. Therefore, the plan 2 is better than my algorithm in the two test queries.

However, the execution time of plan 2 does not include the time for comparing the size of two join query participating tables. Plan 1 and plan 2 are the sequential query processing plans. For the test join queries, plan 2 is much better than plan 1 because plan

2 transfers the smaller table while plan 1 transfers the larger table. That does mean plan 2 is always better than plan 1. It is necessary for the distributed database system to seek which sequential query processing plan is better before database system executes the join query. The total response time of plan 2 is equal to the execution time measured in my test plus the time for comparing the size of two source tables. After added the time for data size comparison, the total response time of plan 2 is probably greater than my algorithm in figure 28 and similar to my algorithm in figure 29. Because I do not have data on the time cost of comparing two source tables, I can not prove that the execution time of my algorithm is less than the total response time of plan 2 based on data.

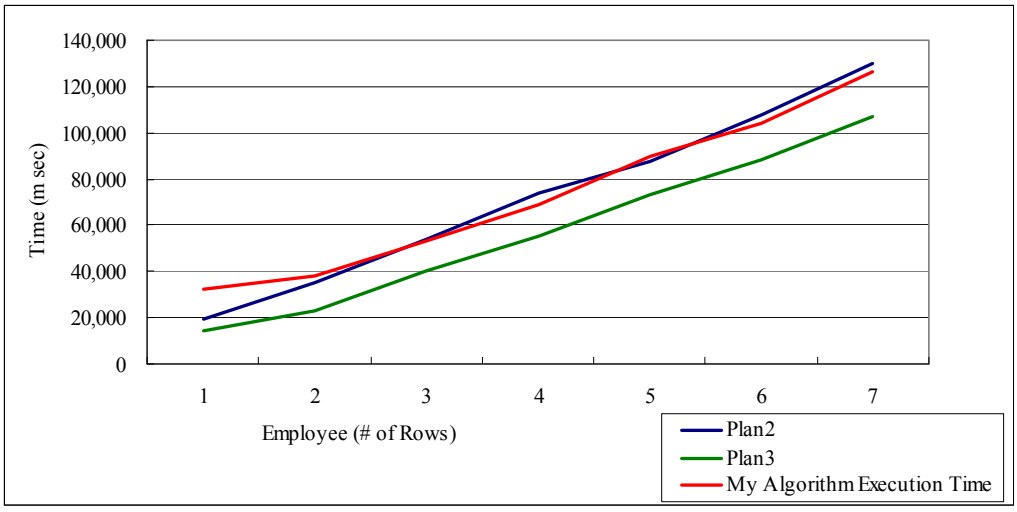


Figure 32 - Execution time of my algorithm, plan 2, and plan 3 when selecting 150 department records

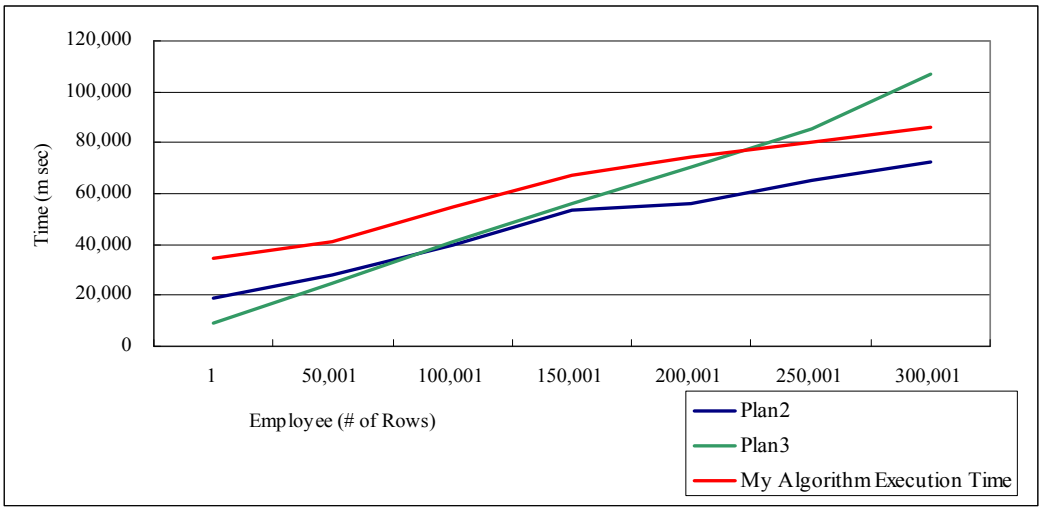


Figure 33 - Execution time of my algorithm, plan 2, and plan 3 when selecting 50 department records

Figure 32 shows that plan 3 is better than my algorithm when the join query selects 150 rows from the department table. Figure 33 shows that plan 3 is better than my algorithm when the join query selects the small numbers of records from the employee table, but plan 3 takes more execution time than my algorithm when join query selects the larger numbers of records from the employee table.

In conclusion, my algorithm does not execute the two test join queries in the shortest time because my algorithm spends time on predicting the best plan for the join query. However, my algorithm can predict the best plan most of the time and avoid executing the join query in the worst plan so that saves the time on join query execution in a distributed database system.

Chapter 6 Conclusion

In this report, I have presented a service-oriented architecture including a set of web services implementing a join query in the distributed database. The system can perform the join query effectively through the interaction of the system components: join query recognizer, optimizer, and executor and metadata database.

I also have proposed a join query optimization algorithm which predicts the best execution plan for a join query which retrieves data from two remote computers. My algorithm estimates and compares the total response time of three possible join query execution plans. The total response time of each execution plan is impacted by the size of transmission data and speed of transmission. The size of transmission is determined by the type of join query. The speed of transmission is calculated with the training results using the Least Square algorithm.

My system development follows the Unified Process model. The project was developed in four static phases: inception, elaboration, construction, and transition. The functions of system were implemented iteratively in each phase. Although the Unified Process model provides me a framework to complete my project in an orderly manner, the time cost of my project was larger than what I planned because of a longer system test. In the system test, new factors which impacted the system performance were found. The system was changed according to the new findings, and a test on the revised system was implemented again. Also, it is very time-consuming to test the data transmission. The response time for transferring a large amount of data is very long, and it is necessary to run the data transmission many times to derive an accurate average.

In the data transmission test, I discovered that the total response time of data transmission in my system comprises of the time for querying, transferring and inserting.

The querying time is short compared to the time for transferring and inserting, but it is important to ensure that the size of querying data does not exceed the system memory limitation. The time for transferring data is significant when the transmission speed is slow and the size of transmission data is large. Although I improved the performance of inserting data by using batch insertion, the time for inserting data is very long due to the limitation of local database management system. Moreover, the test results of data transmission shows that the transmission speed is linear and has a strong correlation with respect to the amount of data transmitted.

The function of parallel query processing was tested. The response time of parallel query processing on a multi-core computer is less than the sum of the response times of two sequential data transmissions and approximately equal to the maximum value of the response times of the sequential data transmissions.

In the test of the join query performance, I proved that the neither parallel query processing nor the sequential query processing are the best plan in all cases. The parallel query processing is better than sequential query processing when the size of the join query result is equal to or larger than the maximum of the sizes of two source tables. The sequential query processing is faster than parallel query processing when size of the join query result is relatively small compared to the source tables. In addition, the transmission speed is a factor which the system must consider to predict the best plan for a join query.

Finally, I tested if my join query optimization system can predict the best plan for a join query. My algorithm can estimate the size of join query result approximately based on the average size of a number of rows and the type of join query.

6.1 Limitation

After I finished all the tests, I found that the limitations of my algorithm and system are as follows.

1. There is error associated with estimating the size of the join query result. In order to increase the efficiency of my system, my algorithm does not count the actual size of source table row by row. Instead, my system estimates the size by calculating the average size of the first 50,000 rows in the source table. If the sizes of the rows are significantly different, the estimated size of the result will be significantly incorrect.
2. There is error associated with estimating the execution and transmission times. Although the transmission time is strongly correlated to the number of bytes transmitted, the actual execution time is different from the estimated time because the network speed is not constant. If the change of speed is significant, the error in the estimated time will be large. Therefore, the error rate of my system is high when the data transferred across an instable network.
3. My algorithm did not include the time for the web services initialization into the estimated total response time. The time for the web services initialization is significant to the total response time when the time of data transmission is short. As a result, the error rate of the estimated response time is high when the join query transfers a small amount of data.
4. Inserting data is time-consuming in my system. In my system, data are stored in MS SQL Server. It takes a long time to insert a large amount of data into

MS SQL Server. The long inserting time decreases the efficiency of data transmission in my system.

6.2 Future Work

In the future, the performance of my join query optimization algorithm can be improved by finishing these works:

1. Verify the right method to estimate the average size of each row. We can use a different number of rows to estimate the average row size instead of 50,000 rows. Also, there are other ways to select the sample rows. For example, selecting the last N rows, selecting a sample in every Nth row, selecting N rows randomly, and so on. We can seek a fast and accurate method to estimate the size of transmission data by testing those different methods.
2. Implement the metadata database. When my system is used in a real business environment, the total response time and the transmission data size of each join query execution could be stored in the metadata database. My system would update the transmission and execution speeds using the latest join query execution information. These updated transmission and execution speeds can improve the accuracy of the estimated response time of the join query execution in a changing network condition.
3. Include the time for the web services initialization into the estimated total response time. This work can improve the accuracy of the estimated response time of the join query execution when the query transfers a small amount of data.
4. Research the methods to inserting data. Inserting data into the database is a significant component of the time to perform the distributed join

query. Further study is needed to investigate ways to improve the performance of insertion.

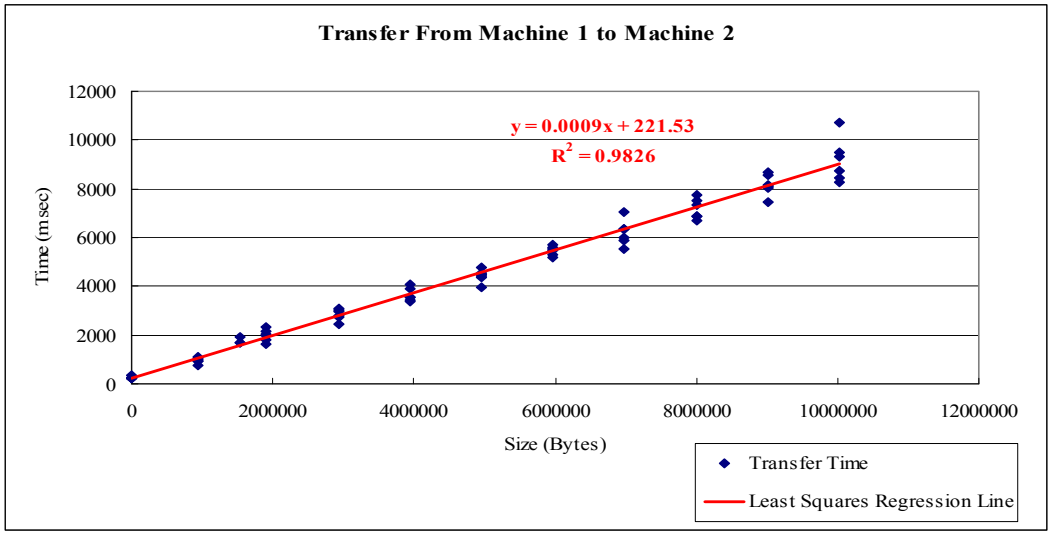
Furthermore, my method can also be expanded to predict the best execution plan for the more complex join query which involves more than three sites. That means the join optimization algorithm can handle all kinds of join queries. Also, the functions of the parallel query processing can be used to improve the performance on selecting data from multiple sites.

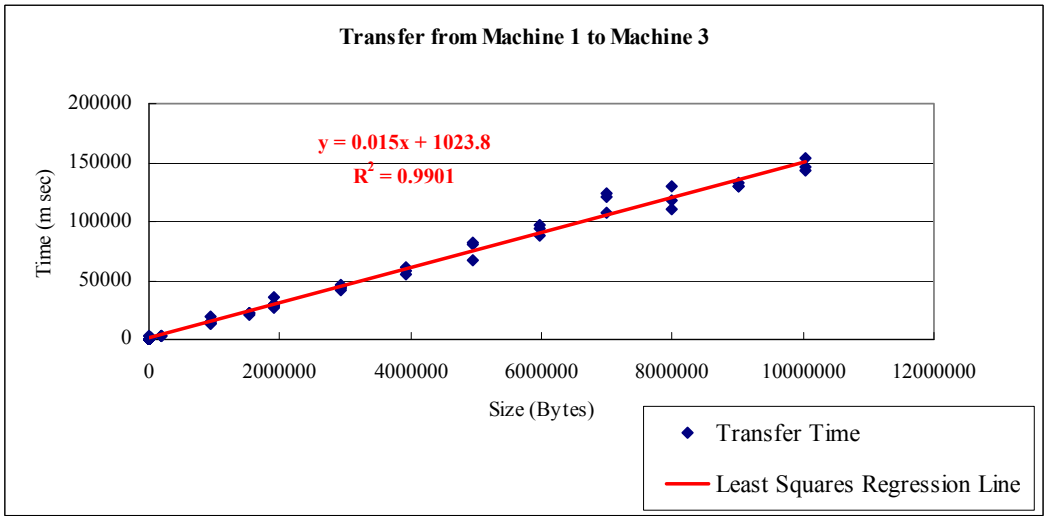
References

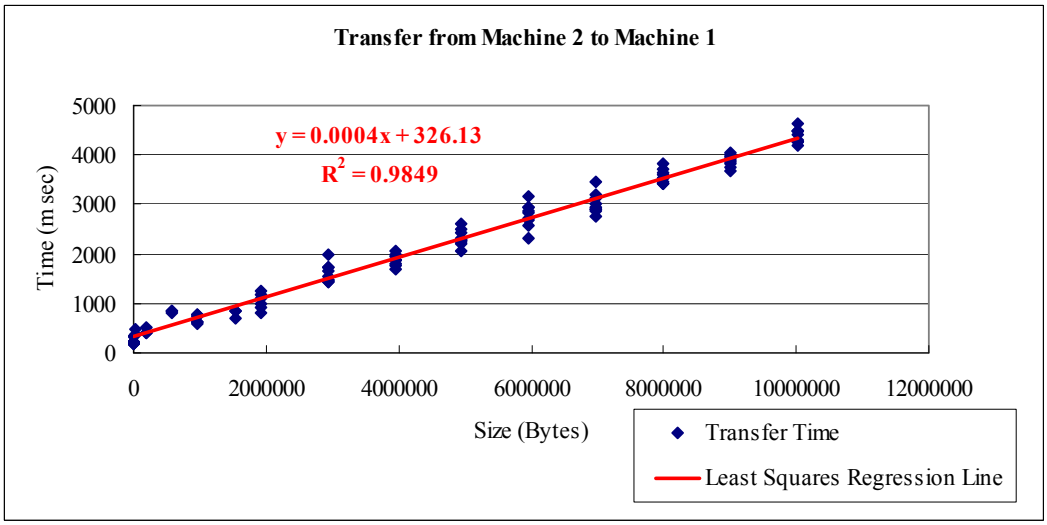
1. Pascal Wehrle, Maryvonne Miquel, Anne Tchounikine, A Grid Services-Oriented Architecture for Efficient Operation of Distributed Data Warehouses on Globus
2. M. Tamer Özsu, Patrick Valduriez, Chapter 1 Introduction, Principles of Distributed Database Third Edition
3. Reza Ghaemi, Amin Milani Fard, Hamid Tabatabaee, and Mahdi Sadeghizadeh, 2008, Evolutionary Query Optimization for Heterogeneous Distributed Database Systems, World Academy of Science, Engineering and Technology 43 2008
4. M. Tamer Özsu, Patrick Valduriez, Section 14.3 Parallel Query Processing, Principles of Distributed Database Third Edition
5. Robert Viera, Professional SQL Server 2005 Programming, Wiley Publishing, Inc.
6. SAKTI PRAMANIK AND DAVID VINEYARD, 1988,Optimizing Join Queries in Distributed Databases, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 14, NO. 9, SEPTEMBER, 1988
7. SINGH, M. P. AND HUHNS, M. N. 2005. Service-Oriented Computing: Semantics, Processes, Agents. Wiley, New York.
8. Gottschalk Et Al. Introduction to Web services architecture, IBM systems journal, Vol 41, No 2, 2002
9. A.H. Anderson. Domain-independent, Compos able Web services policy assertions. In Proc. of the 7th IEEE Int'l Workshop on Policies for Distributed Systems and Networks, pages 1490152, IEEE CS 2006.
10. S. Bajaj, et al. Web Services Policy Framework 1.2 Spec. BEA, IBM, Microsoft, SAP, Sonic, VeriSign, Mar. 2006.

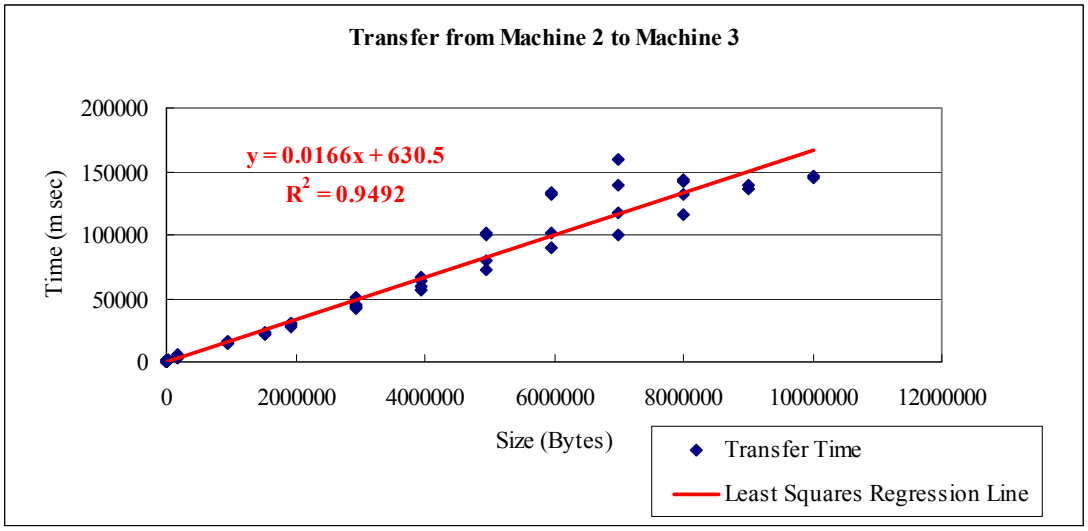
11. I. Foster. Globus toolkit version 4: Software for service-oriented systems. Journal of Computational Science and Technology, 21(4):523–530, 2006
12. JDBC: http://java.sun.com/javase/technologies/database/
13. Grady Booch, Ivar Jacobson, and James Rumbaugh, *Unified Modeling Language 1*.3, White paper, Rational Software Corp., 1998.
14. Barry W. Boehm, Anchoring the Software Process, IEEE Software, 13, 4, July 1996, pp. 73-82.

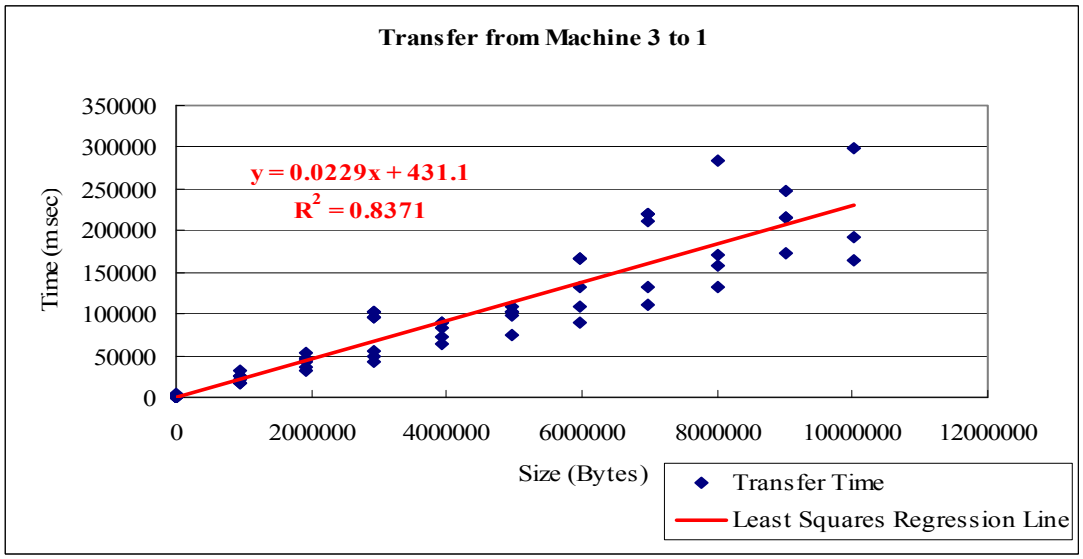
Appendix A - Data Transmission Training Results

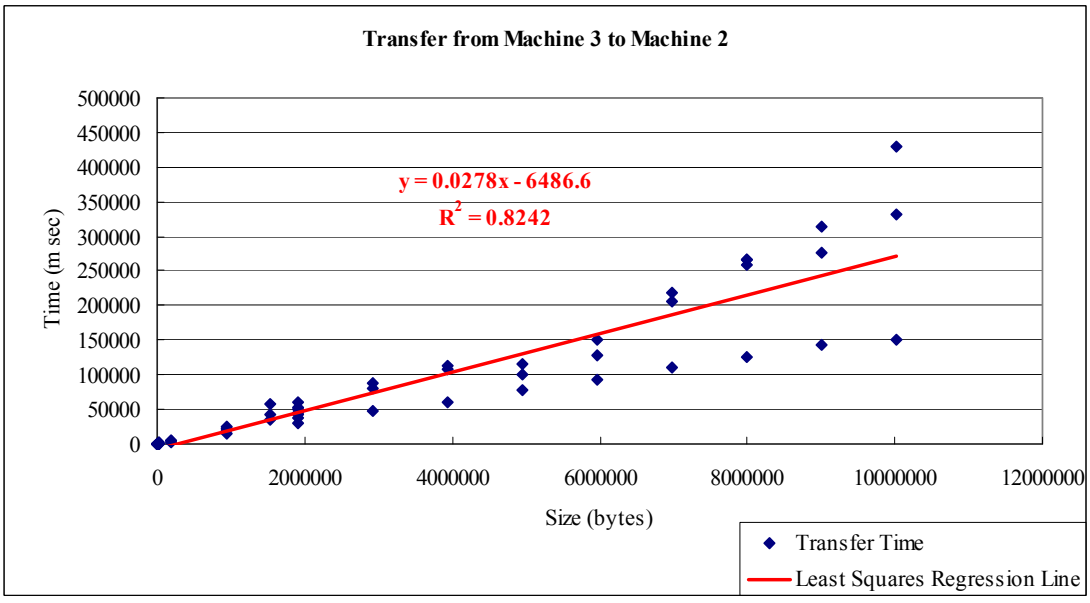


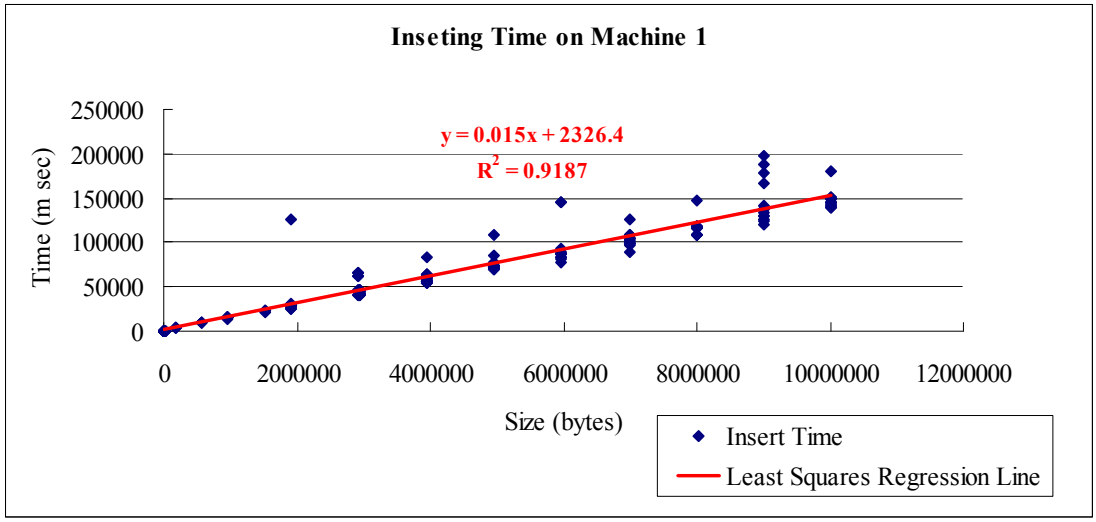


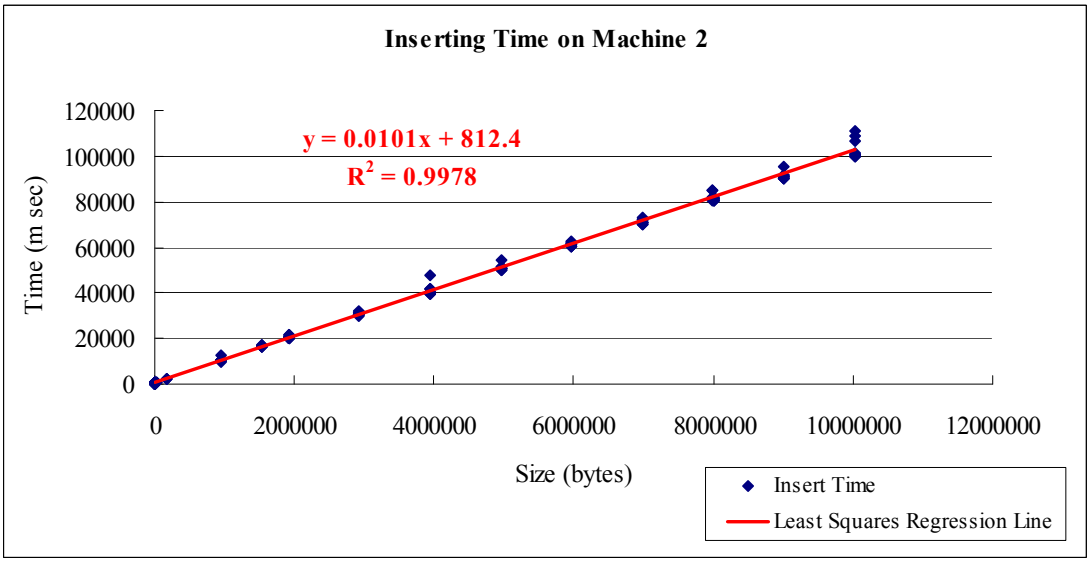


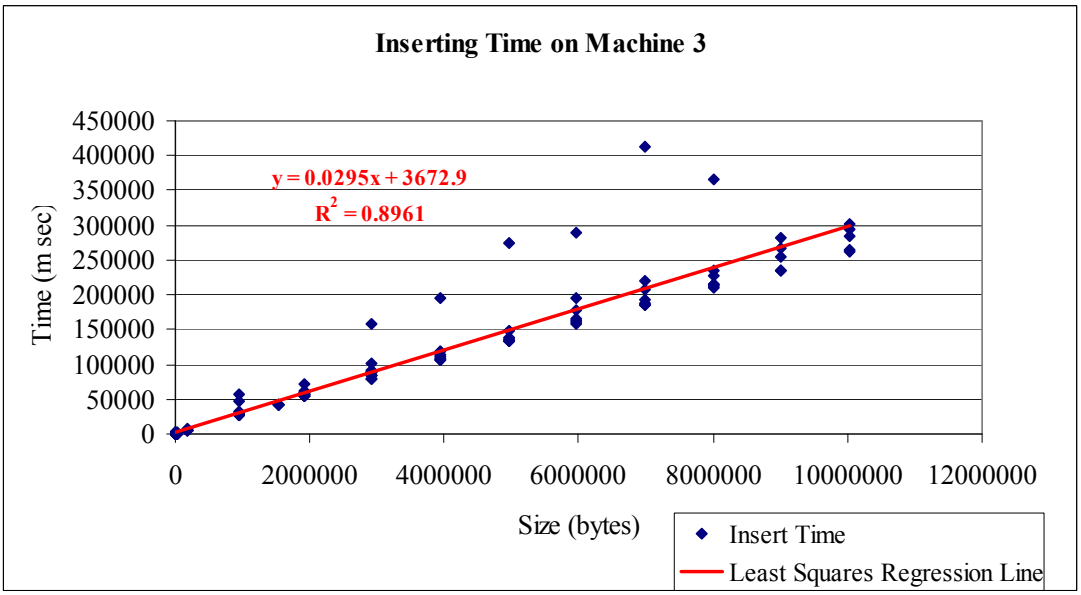

Transfer from Machine 2 to Machine 3
y = 0.0166x + 630.5
R2 = 0.9492
Time (m sec)
Size (Bytes)
Transfer Time
Least Squares Regression Line

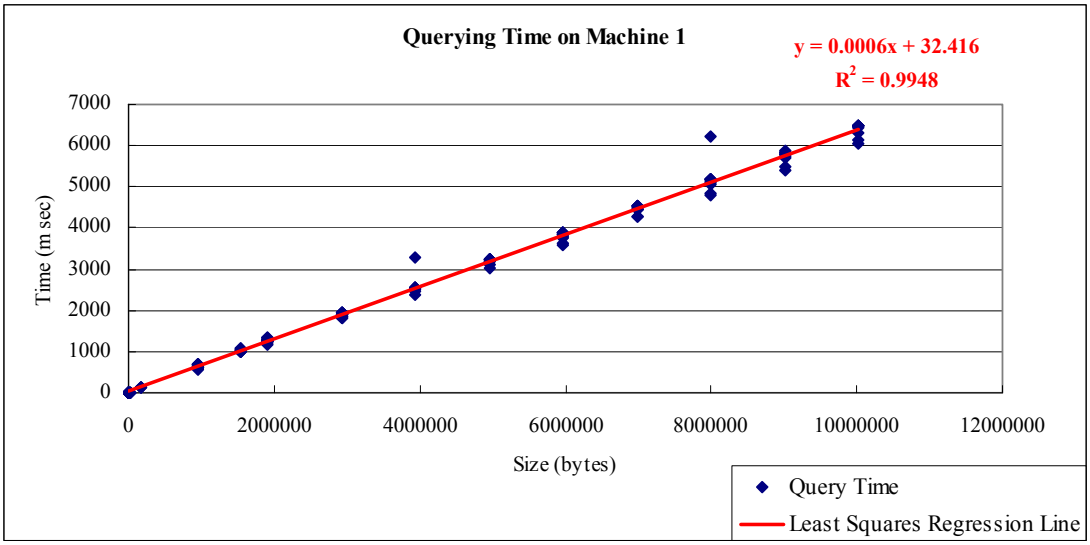

Transfer from Machine 3 to 1
y = 0.0229x + 431.1
R2 = 0.8371
Time (msec)
Size (Bytes)
Transfer Time
Least Squares Regression Line

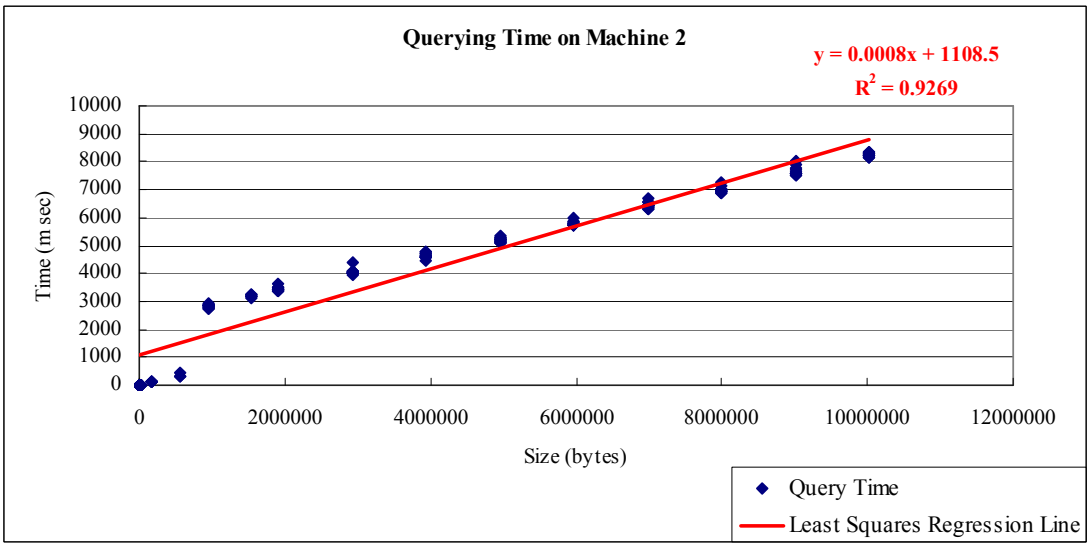

Transfer from Machine 3 to Machine 2
y = 0.0278x - 6486.6
R2 = 0.8242
Time (m sec)
Size (bytes)
Transfer Time
Least Squares Regression Line

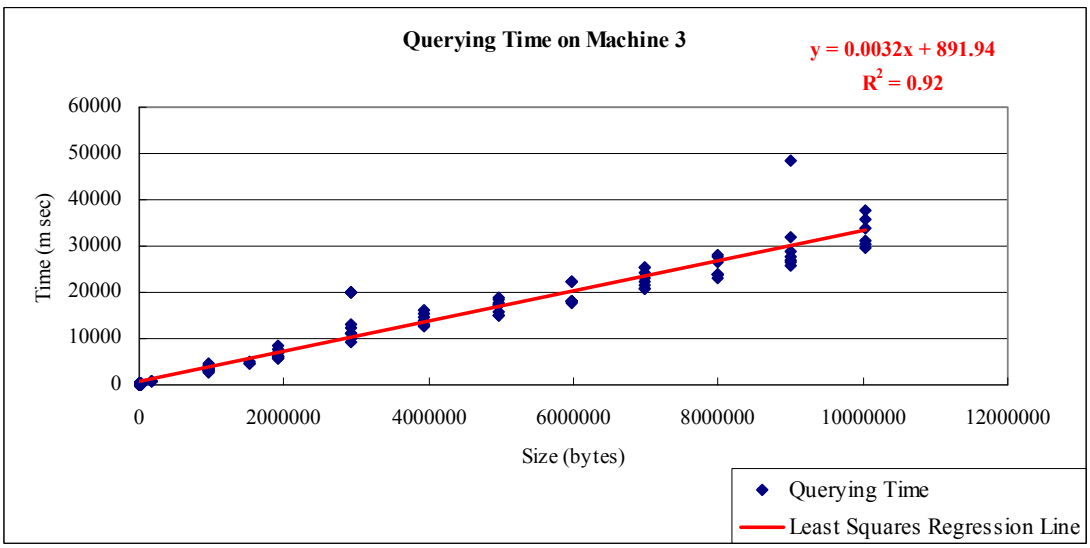

Inseting Time on Machine 1
y = 0.015x + 2326.4
R2 = 0.9187
Time (m sec)
250000
200000
150000
100000
50000
0
0
2000000
4000000
6000000
8000000
10000000
12000000
Size (bytes)
Insert Time
Least Squares Regression Line


Inserting Time on Machine 2
y = 0.0101x + 812.4
R2 = 0.9978
Time (m sec)
120000
100000
80000
60000
40000
20000
0
0
2000000
4000000
6000000
8000000
10000000
12000000
Size (bytes)
Insert Time
Least Squares Regression Line


Inserting Time on Machine 3
y = 0.0295x + 3672.9
R2 = 0.8961
Time (m sec)
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
0
2000000
4000000
6000000
8000000
10000000
12000000
Size (bytes)
Insert Time
Least Squares Regression Line


Querying Time on Machine 1
y = 0.0006x + 32.416
R2 = 0.9948
Time (m sec)
Size (bytes)
0
1000
2000
3000
4000
5000
6000
7000
0
2000000
4000000
6000000
8000000
10000000
12000000
Query Time
Least Squares Regression Line


Querying Time on Machine 2
y = 0.0008x + 1108.5
R2 = 0.9269
Time (m sec)
Size (bytes)
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
0
2000000
4000000
6000000
8000000
10000000
12000000
Query Time
Least Squares Regression Line


Querying Time on Machine 3
y = 0.0032x + 891.94
R2 = 0.92
Time (m sec)
Size (bytes)
0
10000
20000
30000
40000
50000
60000
0
2000000
4000000
6000000
8000000
10000000
12000000
Querying Time
Least Squares Regression Line

Appendix B - Speed and Startup Time Results

Source Machine	Destination Machine	Transfer Speed (bytes/m sec)	Startup Time (m sec)
1	2	1140	221
1	3	66	1023
2	1	2492	326
2	3	60	630
3	1	43	431
3	2	36	-6486

Training result: Transfer Speed and Startup Time

Machine	Transfer Speed (bytes/m sec)	Startup Time (m sec)
1	66	2326
2	98	812
3	33	3672

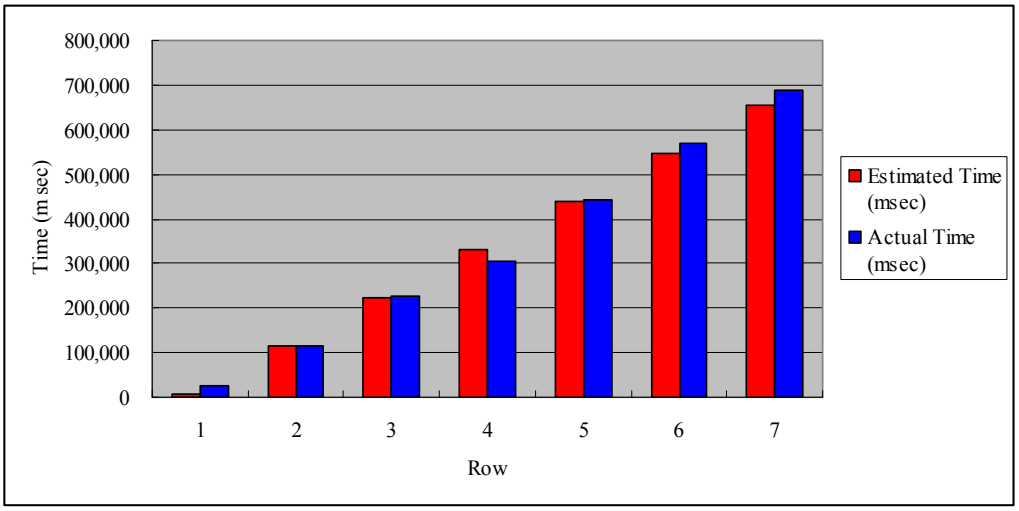
Training result: Insert Speed and Startup Time

Machine	Query Speed (bytes/m sec)	Startup Time (m sec)
1	1575	32
2	1307	1108
3	309	891

Training result: Query Speed and Startup Time

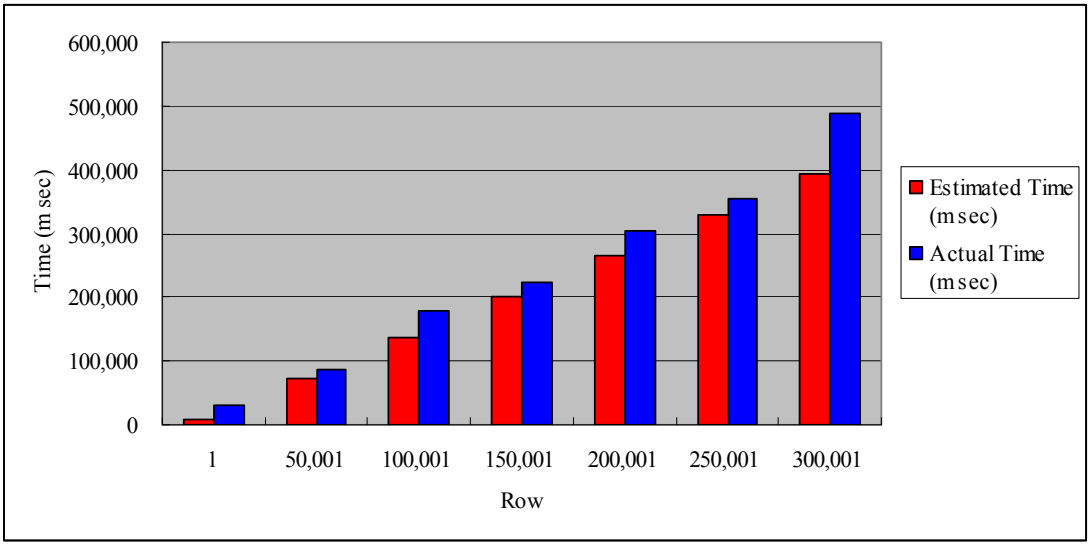
Appendix C - Actual Execution Time and Estimated Time for Plan 1

Department (Row)	Employee (Row)	Estimated Time (m sec)	Actual Time (m sec)	Difference (%)
150	1	9,059	24,639	63.23%
150	50,001	116,569	116,990	0.36%
150	100,001	224,077	227,968	1.71%
150	150,001	331,586	303,690	-9.19%
150	200,001	439,095	444,273	1.17%
150	250,001	546,604	571,092	4.29%
150	300,001	654,113	686,714	4.75%



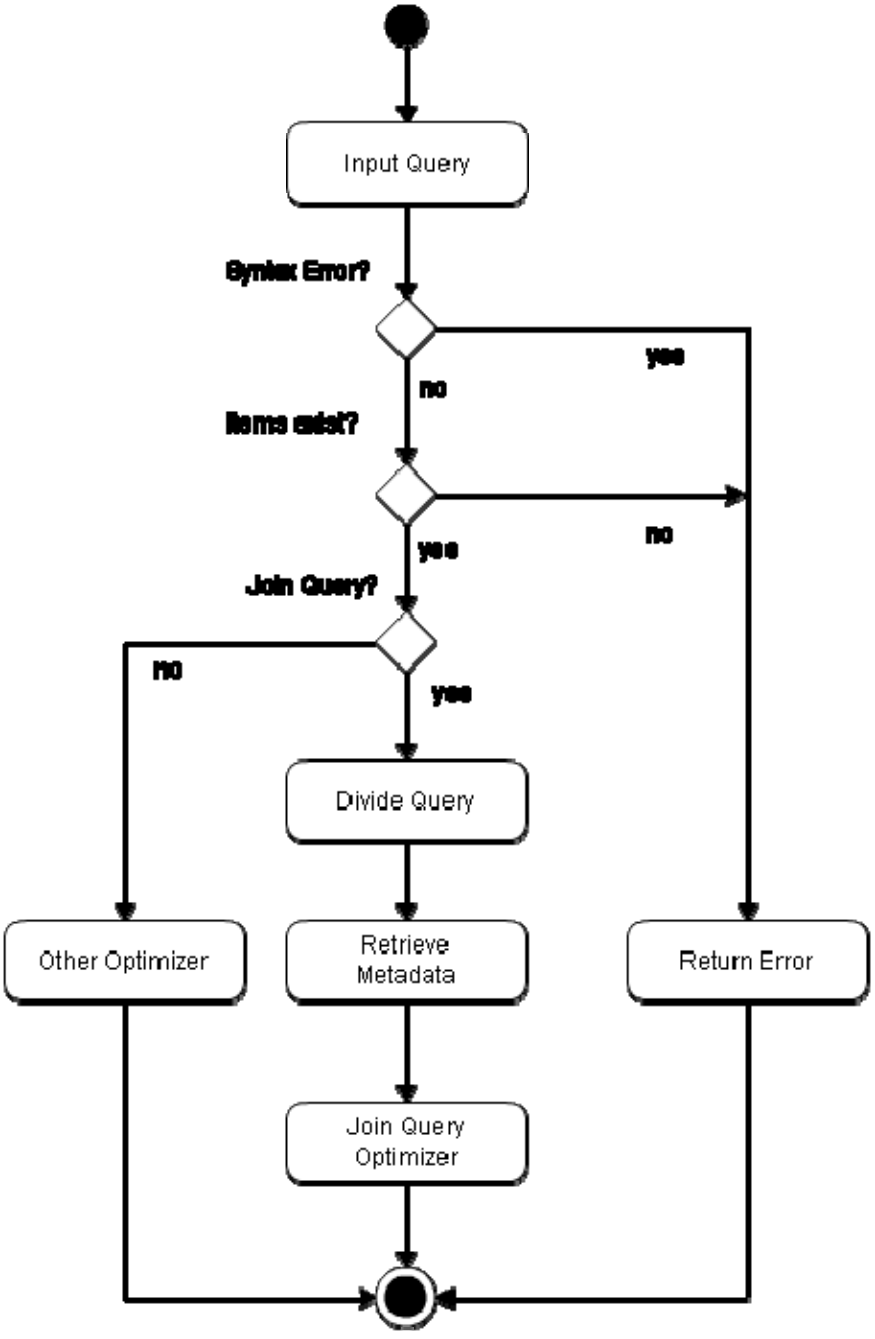
When selects 150 department records

Department (# of Rows)	Employee (# of Rows)	Estimated Time (m sec)	Actual Time (m sec)	Difference (%)
50	1	9,060	30,762	70.55%
50	50,001	73,404	87,820	16.42%
50	100,001	137,754	177,359	22.33%
50	150,001	202,068	223,901	9.75%
50	200,001	266,221	304,692	12.63%
50	250,001	330,355	355,452	7.06%
50	300,001	394,408	489,612	19.44%



When selects 50 department records

Appendix D - Activities Diagram of the Join Query Recognizer



Appendix E – Estimated Size of Join Query Result