Efficient Query Optimization for Distributed Join in Database Federation

by

Di Wang

A Thesis
Submitted to the Faculty of the
Worcester Polytechnic Institute
In partial fulfillment of the requirements for
Degree of Master of Science in
Computer Science
By

_________________________

March 2009

Approved:

_________________________
Professor Murali Mani, Thesis Advisor

_________________________
Professor Elke A. Rundensteiner, Thesis Reader

_________________________
Professor Michael A. Gennert, Head of Department
Abstract

Database federation is one approach to data integration, in which a middleware, called mediator, provides uniform access to a number of heterogeneous data sources. For the mediator, two key components are query rewriter and query optimizer. In this thesis, we focus on the query optimizer part, particularly, on cost-based query optimization for distributed joins over database federation.

One important observation in query optimization over distributed database system is that run-time conditions (namely available buffer size, CPU utilization in machine and network environment) can significantly affect the execution cost of a query plan. However, in existing database federation systems, very few studies have addressed run-time conditions. It is a challenging problem, because usually the mediator is not able to know the run-time conditions of remote sites and considering run-time conditions will bring about extra complexity to the optimizer.

This thesis proposes the Cluster-and-Conquer algorithm for query optimization over database federation while efficiently considering run-time conditions. I firstly propose to view the whole federation as a clustered system, by grouping data sources based on network infrastructure or enterprise boundaries; and then provide each cluster of data sources with its own cluster mediator. The query optimization is divided into two procedures: the global mediator decides inter-cluster operations, and cluster mediators handle the sub queries within the cluster with run-time condition consideration. This algorithm has three-fold benefits. Firstly, the run-time conditions of machines are now available for cluster mediator, because the communication within a cluster is time-efficient. Secondly, each cluster mediator can deal with its own sub query concurrently, so the complexity of processing query plan is decreased. Thirdly, the algorithm outperforms other related approaches in terms of “cost of costing”, because it removes unnecessary inter-cluster operations in the early stage of query plan selection.

I have implemented a prototype data federation system with Cluster-and-Conquer algorithm. The experimental results showed the capabilities and efficiency of our
algorithm and described the target scenarios where the algorithm performs better than other related approaches.
Acknowledgements

This thesis by far is the most significant scientific accomplishment in my life and it would be impossible without people who supported me and believed in me.

Most of all I would like to thank my research advisor, Professor Murali Mani, who gave me comprehensive guidance, unlimited encouragement and wise advices. His tutorial and scientific exploration and his charming personalities inspired me in more ways than I can count. I remember Jan 17 2007, the date I sent the first email to Professor Mani, from then he gave me the great opportunity to change my life. Two years before when I was a new graduate student who had very limited knowledge of database, Professor Mani took time to teach me every piece of DBMS principles with incredible patience; when I tried to find the first few research topics, he taught me the way to start research and gave me practical advice whenever I needed; when I got stuck in my study and/or research, he was always supporting me. It was a great privilege to work with him in my graduate study.

I would also like to express my gratitude to my thesis reader Professor Elke A. Rundensteiner for everything she has done to make this thesis possible.

I am grateful for all of my professors at WPI. Thank them for all the knowledge, inspiration and help.

I have learned a great deal from my fellow graduate students and friends in DSRG. They all deserve my gratitude for their advice, laughs and kindness.

A special gratitude goes to my best friend Zhe Zhou, for his unconditional support.

My deepest gratitude goes to my Mom. I know she is always caring about me from the heaven.
Contents

Abstract ........................................................................................................................................... ii
Acknowledgements ....................................................................................................................... iv
List of Figures ................................................................................................................................. vii

Chapter 1 Introduction ................................................................................................................ 1
  1.1 Data Integration ...................................................................................................................... 1
  1.2 Database Federation .............................................................................................................. 3
  1.3 Considering Run-time Conditions in Query Optimization ................................................. 5
  1.4 Existing Algorithms Considering Run-time Conditions ..................................................... 6
  1.5 Cluster and Conquer Algorithm ............................................................................................ 8

Chapter 2 Related Works .......................................................................................................... 11

Chapter 3 Preliminary ................................................................................................................. 14
  3.1 Cost Model and Optimization Goal ....................................................................................... 14
  3.2 Parallelism and Pipelining Restrictions ............................................................................... 16
  3.3 Assumptions and Other Restrictions .................................................................................... 16

Chapter 4 Typical System Architecture and Data Structure ................................................ 18
  4.1 Typical Architecture of Database Federation System Overview .................................... 18
  4.2 Typical Data Structures and Statistics .............................................................................. 19
    4.2.1 Query Execution Plan ................................................................................................... 20
    4.2.2 Statistics ...................................................................................................................... 21

Chapter 5 Cluster-and-Conquer Algorithm ........................................................................... 22
  5.1 Motivation ............................................................................................................................ 22
5.2 Key Observation ........................................................................................................................................... 23
5.3 Cluster-and-Conquer Algorithm .................................................................................................................. 26
  5.3.1 Proposed System Architecture ................................................................................................................ 26
  5.3.2 Proposed Data Structure and Optimization Process .............................................................................. 28
  5.3.3 Algorithm Lists and Running Example .................................................................................................. 30
5.4 Cost Model Analysis ...................................................................................................................................... 35
5.5 Algorithm Design Analysis ........................................................................................................................ 36
  5.5.1 Query Plan Search Space ..................................................................................................................... 36
  5.5.2 Run-time Condition Consideration and Overhead .............................................................................. 37

Chapter 6 Experimental Evaluation ............................................................................................................... 38
  6.1 Experimental Setup .................................................................................................................................... 38
  6.2 Overview of Comparable Approaches ..................................................................................................... 41
  6.3 Varying Available Buffers ....................................................................................................................... 42
  6.4 Varying Network Factors ......................................................................................................................... 45
  6.5 Cost of Optimization over Size of Query ................................................................................................. 46

Chapter 7 Conclusion and Future Work ......................................................................................................... 49
  7.1 Conclusion .................................................................................................................................................. 49
  7.2 Future Work .............................................................................................................................................. 50
References .......................................................................................................................................................... 52
List of Figures

Figure 1 Application-specific solution for data integration ........................................ 2
Figure 2 Typical Database Federation Instance .............................................................. 4
Figure 3 Execution plans for Query 1.1 ........................................................................ 6
Figure 4 Typical Architecture of Database Federation .................................................... 19
Figure 5 Centralized plan for Query 4.1 ....................................................................... 20
Figure 6 Distributed plan for Query 4.1 ....................................................................... 21
Figure 7 A global-scale database federation overview .................................................. 24
Figure 8 A federated database with data sources from multiple companies .................. 25
Figure 9 Viewing the federation as a set of clusters ...................................................... 26
Figure 10 Abstract of the clusters from Figure 9 ............................................................ 26
Figure 11 Proposed System Architecture with Multi-level Mediators ......................... 27
Figure 12 Clustered View of Tables Used by Global Optimizer .................................... 28
Figure 13 Static Clustered Plan Based on the Clustered View in Figure 12 .................. 29
Figure 14 Distributed Clustered Plan Produced According to Figure 13 ...................... 29
Figure 15 Physical Execution Plan for Cluster 1 ............................................................ 30
Figure 16. Table Schemas for Query 5.1 ....................................................................... 33
Figure 17: Physical Setting in Execution of Query 5.1 .................................................... 34
Figure 18 Static Clustered Plan Generated by Algorithm1 for Query 5.1 ...................... 34
Figure 19 Execution Plans Generated by Cluster Optimizers with Algorithm2 .......... 34
Figure 20 Basic Experiment Architecture Setup ........................................................... 39
Figure 21: Running Time of Query 6.1 with Varying Buffer Size .................................. 43
Figure 22: Running Time of Query 6.1 with Varying Buffer Size .................................. 44
Figure 23: Running Time of Query 6.1 with Varying Buffer Size .................................. 44
Figure 24 Running Time of Query 6.1 with Varying Network Factors

Figure 25 The Cost of Three Optimization Algorithms with Varying Query Size and Varying Network Factors
Chapter 1

Introduction

In this section we firstly introduce data integration technologies and its research thrusts. And then we introduce database federation, including its general definition and key ideas. We will further discuss the query optimization in database federation, which is the research topic in this thesis. We will finally give a rough description of our proposed algorithm.

1.1 Data Integration

Data integration has been, and continues to be, an active research topic, because it is crucial for large enterprises that own a multitude of data sources, for groups that were formed by merging several originally different companies, for progress in large-scale scientific projects, where data sets are being produced independently by multiple researchers, for better cooperation among business partners, and for good search quality across the millions of data sources on the World Wide Web.

In a large enterprise, it is almost inevitable that different parts of the organization will use different systems to produce, store, and search their data. Yet, it is only by integrating the information from these various systems that the enterprise can realize the full value of the data contained [18].

In the finance industry, mergers are an almost commonplace occurrence. After the merger, the new company needs to be able to access the customer information from both sets of
data stores, to analyze its new portfolio using existing and new applications, and, to use the combined resources of both institutions through a common interface [1].

In addition, today’s companies have been interested in combining data with its business partners. Because of the continuous creation of business relationships and partnerships, many companies require processing of data that belong to more than one institute [17].

Besides the ubiquitous need for data integration in business world, there is a growing interest in the scientific community to allow disparate groups of researchers to share resources consisting of both data collections and programs [2, 3].

Also the World Wide Web is witnessing the need to deal with vast heterogeneous collections of data sources. Improving search by inter-operating among data sources poses one of the greatest challenges [5].

There are many mechanisms for integrating data. These include application-specific solutions, data warehousing and database federation. The application-specific solutions provide special-purpose applications that access data sources of interest directly, and then combine data via the application itself, as shown in Figure 1. This approach always works, but it is expensive, which requires skillful programmers with good knowledge of every data sources. Also it is so fragile that changes to the underlying data sources may all ask for changes to the application.

Figure 1 Application-specific solution for data integration
Data warehousing, by contrast, provides users with a uniform interface to combine and manipulate their data. A data warehouse is built by loading data from one or more data sources into a newly defined uniform schema in a database [19]. The data are often cleaned and transformed in the load process. Changes in the underlying sources may cause changes to the load process, but the part of the application over the uniform schema is protected.

However, data warehousing may not always be a solution. For example, it is possible that moving data from their original location is not feasible or forbidden [1]. Also data warehouse comes with its own maintenance and implementation costs. Database federation is another solution which has all the benefits of data warehousing without necessarily moving any of the data.

1.2 Database Federation

Database federation is one approach to data integration in which middleware, called mediator, provides uniform access to a number of heterogeneous data sources. The key performance advantage offered by database federation is the ability to efficiently combine data from multiple sources in a single query statement. The data sources are federated into the unified mediator. User can submit a query that access data from multiple sources, joining and restricting, aggregating and analyzing the data at will, without knowing what exactly the sources are. To achieve this transparency, a wrapper is built for each data source, which encapsulates the data source, and mediates between data source and the mediator. A typical database federation instance is shown in Figure 2.
For the mediator, two key components are query rewriter and cost-based optimizer. The query rewriter can aggressively rewrite a user’s query into a semantically equivalent form that can be more efficiently executed across multiple sources. The cost-based optimizer can search a large space of feasible execution plans and choose an optimal plan based on the cost metric.

For wrappers, the interfaces are crucial, because wrappers are responsible for managing the diversity of data sources [21]. Below a wrapper, each data source has its own data model, schema, programming interfaces, and query capability. Wrappers are written to standardize how information in data sources is described and accessed. Many middleware systems are using wrapper architecture and providing quick and flexible way to build wrappers, e.g. Garlic [20], IBM’s DB2 [1].

Most research thrusts of database federation can be categorized into two areas: query capability technologies and query optimization. Researchers working on query capability technologies focus on such topics as answering queries using views (they consider data sources or wrappers as views) [22], containment algorithms for conjunctive queries [23], schema mapping and matching [24]. Researchers interested in query optimization over
database federation have been working on cost-based optimization [8], adaptive query processing [11] and query parallelization technologies [16].

As seen from its typical architecture, database federation is a distributed system by nature. Thus, in fact, many query optimization work here are closely related to early optimization techniques developed for the distributed database systems, e.g. R*[25], Mariposa [9, 10].

In this work, we focus on cost-based optimization for distributed joins over database federation and propose a new approach to build the optimal plans in specific federated environments.

1.3 Considering Run-time Conditions in Query Optimization

When it comes to cost-based optimization over distributed data system, it is a fairly straightforward observation that run-time conditions can significantly affect the execution cost of a query plan. One example is that, available buffer size in the system will determine the number of “runs” in sort operation, and affect such operations as nest-loop join, etc. Another example is CPU utilization, which will determine the possible speedup of query execution. However most of the existing database federation optimizers fail to take run-time conditions into account. When measuring the cost of a candidate query plan, often such optimizers in the mediator consider the costs of operators in each site as static values [2, 4, 8, 9, 20]. Hence, a query plan has a constant estimated cost, and an input query has a fixed output ‘optimal’ execution plan consequently. On the other hand, it is common that the optimizer in the mediator is not aware of the run-time system conditions of remote sites [3, 5, 11], mainly because after significant transfer delay, the run-time condition values are not accurate any more.

If run-time conditions are not considered, the query plan that was optimal at optimization time may have bad performance at run time. An example is as follow:

Query 1.1: Natural Join three relations, R1, R2, R3.
Physical Locations: R1 is in data source M1, R2 is in data source M2, R3 is in data source M3.

Catalogs and Estimation: B(R1) < B(R2) < B(R3), B(R1 join R2) < B(R3)

B(R) means the size of relation R in terms of number of blocks. Given the above information, a common distributed query optimization will enumerate several execution plans, and the following three are possible optimal plan candidates among them:

![Execution plans for Query 1.1](image)

These three plans have the same join ordering, while the operation locations differ. We assume that the optimization cannot consider run-time conditions of data sources, which is what most existing optimizers in database federation do. However, the correctness of the choice depends on real run-time conditions. As shown in Table 1, in each run of the example query, each data source has different run-time conditions (CPU utility and available buffer size are considered here), and different optimal plan is computed based on the classical algorithms described in [35]. No matter which plan is chosen by the optimizer, since it is constant, very likely, it is not the optimal one against certain run-time conditions.

<table>
<thead>
<tr>
<th>Run</th>
<th>CPU Utility</th>
<th>Available Buffer</th>
<th>Optimal Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M1</td>
<td>M2</td>
<td>M3</td>
</tr>
<tr>
<td>1</td>
<td>25%</td>
<td>25%</td>
<td>25%</td>
</tr>
<tr>
<td>2</td>
<td>75%</td>
<td>10%</td>
<td>25%</td>
</tr>
<tr>
<td>3</td>
<td>50%</td>
<td>50%</td>
<td>15%</td>
</tr>
</tbody>
</table>

Table 1 Run-time Conditions do affect optimal plan

1.4 Existing Algorithms Considering Run-time Conditions
In other data management systems than database federation, run-time conditions have been studied by several works.

*Parametric query optimization* [14] attempts to identify several execution plans, each one of which is optimal for a subset of all possible values of the run-time parameters. Since the number of values (or the combinations of values) of parameters is large, the optimizer has to explore a huge set of alternative plans. This approach hence highly depends on an economical exploration algorithm (randomized algorithm is what they used). As long as the query is likely to be executed many times, and the result of optimal plan selection for specific run-time conditions can be cached, thus the cost of optimization would be a trade-off. However, even though the overhead of producing multiple plans for one query can be acceptable in centralized database system, the size of plan space is prohibitive in distributed database systems. By considering the site selection (i.e. to assign which machine to which operation) in addition to the algebraic transformation and physical methods selection, the optimization is likely to become a very costly process.

The XPRS project proposes a *two-phase algorithm* for multi-user parallel databases [15]. Namely, in the first phase, which is performed at compile time, only sequential query execution plans are considered. In the second phase, which is performed at run time, it finds the optimal parallelization of the best sequential plan chosen in the first phase, with the information of run time parameters. This algorithm has been further extended in distributed environment [12]. The second phase then requires the exhaustive search of all possible site selection. This simple algorithm surprisingly performs well, as long as it is assumed that communication cost forms a small fraction of the total cost and that the exhaustive search in the second phase is not very expensive. However, these assumptions cannot generally hold. The cost of data transfer of large size files through a long-haul network can be pretty high. Moreover, if the scale of data sources is large, the number of exhaustive permutations of all sites, for one static plan though, can be huge.
Parametric optimization and two-phase algorithm originally targeted at non-distributed database system (Two-phase algorithm was extent to distributed system later), while in distributed database systems, especially distributed data integration system, very few studies have addressed run-time conditions, which we will discuss further in Chapter 2.

1.5 Cluster and Conquer Algorithm

I propose the *Cluster-and-Conquer* algorithm for query optimization over distributed database federation considering run-time conditions of data sources.

Firstly, we consider all data resources in the database federation as a set of several clusters of sites. This abstraction accords with many real-world facts: 1) many national-scale or global-scale data federations are built on the networks which consist of both broad, LAN paths and narrow, long-haul paths. Hence, a bunch of sites connected via LAN can be viewed as a cluster, or a bunch of sites geographically located within a certain area can be viewed as a cluster; 2) many highly-integrated systems have to access data through a great deal of databases that belong to multiple different organizations. In such a case, the set of databases that belong to the same single organization can be viewed as a cluster.

Secondly, we design two layers of mediators to schedule the query plan cooperatively. The *global mediator* produces a high-level optimal plan over several clusters, in which only inter-cluster operations are determined, while passing inter-cluster queries to corresponding cluster mediators. And then the *cluster mediator*, which virtually resides in each cluster, will deal with the sub-plan passed that only need to access data sites in this cluster. The cluster mediator is responsible to collect run-time parameters from related data sites, and then uses this information while computing the intra-cluster optimal plan.

Obviously this cluster-and-conquer optimization approach decreases the plan search space, because it eliminates numerous plans that unnecessarily join tables across distinct clusters. Considering that only a subset of plans will be fully explored during the
optimization process, we may expect this approach to produce much worse plans than exhaustive algorithm. However, notice that the data sources are clustered based on several essential properties of a database federation, such as: there exist enterprise boundaries, which forbid moving data to other enterprises’ sites; or in a global database federation, data transfer through long-haul paths is pretty costly, while data transfer within a LAN is economical. Hence joining primitive tables across distinct clusters is either infeasible or prohibitive. Moreover, our approach releases the global mediator from the cumbersome work of collecting or estimating all sites’ run-time parameters. Later we will introduce our experiments which show that our approach does perform well.

What is more, having the cluster mediator handle intra-cluster queries has three-fold benefits. Firstly, the communication within a cluster is time-efficient, so the value of run-time parameters collected by the cluster mediator is much fresher than that gathered by the global mediator. Secondly, each cluster mediator can deal with its own query concurrently, which implicatively employs the independent parallelism. Thirdly, the complexity of the centralized optimization of a whole query plan in distributed environment is greatly decreased, since cluster mediators can conquer every piece of less complex sub-plan respectively.

1.6 Contributions

This thesis contributes to the advancement of query optimization over database federation in the following ways:

- I observe and use the clustered feature of large-scale database federations and take advantage of the divide-and-conquer concept in distributed systems.
- I present the need and challenge for considering run-time conditions of data sources and network in the process of query optimization.
- I propose Cluster-and-Conquer algorithm for query optimization of distributed joins over database federation.
- I present an analytical evaluation of Cluster-and-Conquer algorithm in terms of fulfilling the request of run-time condition consideration with little overhead.
I implement Cluster-and-Conquer algorithm on the simulating data federation system from scratch.

I provide experimental evidence that Cluster-and-Conquer algorithm improves the performance when evaluating queries over standard data generated by TPC-H benchmark. Also I provide comparative analysis and experiments with other query optimization algorithms over database federation and verify the efficiency of Cluster-and-Conquer algorithm.

### 1.7 Organization of the Thesis

- This thesis is divided into seven chapters.
- Chapter 1 contains this introduction.
- Chapter 2 contains an overview of past achievement and related work in data integration and data federation, especially the query optimization in those works.
- Chapter 3 contains preliminaries, including basic cost-based query optimization concepts, distributed query processing technologies and the assumptions and restrictions of this work.
- Chapter 4 provides general architecture of database federation, and introduction of typical data structure and optimization process.
- Chapter 5 discusses the Cluster-and-Conquer algorithm, including the motivation and key observation of this algorithm, as well as detailed design and theoretical analysis.
- Chapter 6 provides experimental evaluations of Cluster-and-Conquer algorithm and the conclusions about the efficiency of the algorithm
- Chapter 7 concludes and describes possible extensions of this research work.
Chapter 2

Related Work

Many research projects and a few commercial systems have implemented and evolved the concept of database federation. Pioneering research projects include TSIMMIS at Stanford University [26], which used database concepts to implement “mediator”. TSIMMIS described key components required in typical database federation: component that extract properties from heterogeneous data sources, component that translate information into a common object model, component that combine information from several data sources, and component that answers queries over multiple data sources. The important focuses of TSMMIS are common model design and the capability of browsing and combining objects. Query optimization over the whole system has not been addressed.

Garlic at IBM [27] is the first research project to exploit the full power of a standard relational database (DB2). The wrapper architecture and cross-source query optimization of Garlic are now fundamental components of IBM’s federated database offerings [1]. Later, Garlic developed a complicated framework for cost-based query optimization across sources [8]. However, all cost factors and cost formulas used in the framework are within the context of traditional query optimizer. Features of distributed system such as hardware conditions and run-time conditions of data sources were not studied in Garlic.
Consequently IBM provided series of commercial database federation products, including DB2 DataJoiner and DB2 Information Integrator. IBM InfoSphere Federation Server [28] is its most updated commercial database federation product. As commercial products, these systems focus mainly on heterogeneity support (this allows user to combine data from disparate sources such as popular DBMS as well as special purpose software systems into a single virtual view) and extensibility (the ability to add new data sources dynamically). Regarding cost-based optimization, especially optimization with run-time conditions, very few issues have been addressed in those systems.

Similar to IBM’s Garlic, distributed heterogeneous query processor (DHQP) at Microsoft SQL Server [4] is a middleware system design to integrate data from heterogeneous data sources. DHQP in SQL Server adopted the classical mediator architecture. First, OLE DB data access interfaces, a set of industry standard APIs, enable the exposure of data source capabilities from many relational and non-relational data sources. Second, the DHQP is built in to the relational optimizer and execution engine of SQL Server, which serves as a mediator. Its main distinction from IBM’s Garlic is that, DHQP relieves the responsibility of cost functions and optimization rules, which are required for Garlic’s wrapper, from OLE DB data access interfaces. Queries in DHQP in fact are mainly processed with the cost-based algebraic transformations and execution strategies available in centralized SQL Server. One controversial feature of DHQP in SQL Server is that query optimization is processed in a centralized manner, which may incur incorrectness in a distributed system. Moreover, the optimizer embedded in SQL Server has no way to know the runtime conditions of data sources.

Mariposa [10] is a distributed database research system, which proposed the use of an economic paradigm. The main idea behind the economic paradigm is to integrate the underlying data sources into a computational economy that captures the autonomous nature of various sites in the federation. A significant goal of Mariposa was to demonstrate the global efficiency of this economic paradigm. In terms of distributed load balancing, the “global efficiency” is closely related to the reason why we need to consider run-time condition as discussed in Section 1. 3. However, the paradigm is built
on the assumption that each site has total local autonomy to determine the cost to be reported for an option, and can take into account factors such as resource consumptions and hard-ware conditions. There are a few controversies over this assumption: (1) the fully decoupled costing process without a global coordinator / mediator cannot ensure quality of query answering; (2) the requirements for data sources that want to join in the system will be high.

Besides integrating only data, several mediator systems support integrating both data and functions (a.k.a. programs). LeSelect [29] is a distributed data integration system with database federation architecture, which allows users to publish their data and functions. To model data sources including functions, LeSelect used the model of *table with binding patterns*, which was demonstrated to be able to naturally model both data and functions. In this scenario, cost model is difficult to define for both functions and data. In fact, no cost-based optimization is used in the system, which could be a defect in many cases.

Most recently, *Web Service Management Systems* (WSMS) [30, 31] have taken advantage of data integration and mediator technologies. WSMS typically has the similar architecture as mediator system: Web Services can be viewed as one type of data sources, and Web Service adaptor serves as the ‘wrapper’ between every Web Service and the management system. Queries over multiple Web Services can be answered by using similar mechanism as combining data from multiple data sources. Query optimization in WSMS mainly focuses on arranging a query’s web service calls into a pipelined execution plan that optimally exploits parallelism among web services. Still, hardware and run-time conditions are not taken into account in the system.
Chapter 3

Preliminary

In this chapter, I first define the cost model used in this thesis, and then describe the assumptions and restrictions in the system.

3.1 Cost Model and Optimization Goal

Generally speaking, the overall performance goal of a database federation is to obtain increased throughput and decreased response time in a multiuser environment. In this thesis, we consider both the total resources consumed and the response time.

Firstly, we define the cost model for executing query in the non-parallel manner. Given a distributed join operation over $n$ data sources (in this work one data source refers to one machine, so “data source” and “machine” are exchangeable terms in the following contents), we define the cost of a query execution plan for the join as:

\[
\text{Cost}_{\text{NonParallel}} \triangleq w_1 * \sum_{i=1}^{n} Rsrcs_i + w_2 * \sum_{j=i+1}^{n} \sum_{i=1}^{n-1} Trans\_Data_{ij} + w_3 * Resp\_Time
\]

$Rsrcs_i$ is the resource consumed in data source $i$, in terms of number of I/Os. $Trans\_Data_{ij}$ is the size of data transferred from data source $i$ to data source $j$, in terms of megabyte. $Resp\_Time$ is the response time, in milliseconds. Here $w_1, w_2, w_3$ are...
system-specific weighting factors to scale those variables in order to combine them together in meaningful ways.

Secondly, we define the cost model for executing multiple query plans in parallel. Assume that operations within each query plan is executed in non-parallel, thus for each plan we can firstly calculate its \textit{Cost\_NonParallel}. Given \(m\) query plans, the cost of executing those query plans in parallel is defined as the maximum among their \textit{Cost\_NonParallel}, as denoted in the following formula:

\[
\text{Cost\_MParallel} \triangleq \max_{i=1}^{\text{\#plans}} \{\text{Cost\_NonParallel}(\text{plan}_i)\} \quad (1.2)
\]

Thirdly, we define the cost model of a combined query plan, which contains the execution of multiple sub-plans in parallel, as well as contains the execution of non-parallel joins among the results produced by sub-plans. To simplify the cost analysis, we assume that the join ordering of the join operations over the results produced by sub-plans is determined in a greedy algorithm, with greed on smallest number of data transferred. For example, the result produced by \(\text{plan}_i\) is 128MB, and the result is given at data source \(S_i\); the result produced by \(\text{plan}_j\) is 25MB, and the result is given at data source \(S_j\). And then the join between the two results will be execution in two steps: first transfer the result of \(\text{plan}_j\) to \(S_i\), second join the two results at \(S_i\). Thus, given \(m\) sub-plans, the cost of executing a combined query plan is defined as:

\[
\text{Cost\_Combined} \triangleq w_1 \times \text{Cost\_MParallel} + w_2 \times \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \text{Trans\_Data}_{ij} \quad (1.3)
\]

Here \(w_1, w_2\) are system-specific weighting factors to scale those variables in order to combine them together in a meaningful way.

Notice that our cost model is similar to those defined in many existing distributed data management system, but our approach of computing cost is different, in the sense that every variable is computed with consideration of run-time parameters. We will show how the cost computing approach efficiently works with this cost model in following chapters.
3.2 Parallelism and Pipelining Restrictions

Our target database federation is by nature a distributed system, thus we need to consider in what level the query parallelism and pipelining are implemented.

Typically there are three forms of intra-query parallelism [16]:

- **Partitioned parallelism**: A single operator is executed on a set of sites by partitioning its input data set.
- **Pipelined parallelism**: A sequence of operators is executed on a set of sites in a pipelined manner.
- **Independent parallelism**: multiple operators with no pipelining between them can be executed in parallel on a set of sites independent of each other.

The partitioned parallelism is also called intra-operation parallelism, while the other two are called inter-operation parallelism [15]. In this work, we consider only independent parallelism which is a way of inter-operation parallelism, for the following reasons. Firstly, the input data partition is not often feasible among a database federation, because it may not be allowed to move data from their original location. Secondly, in a bushy plan, it is common to have two operations that do not depend on each other’s output, which is ideal to be executed concurrently. To simplify our study, we do not consider the pipelined parallelism, another form of inter-operation parallelism in this work.

3.3 Assumptions and Other Restrictions

Below is the list of current assumptions and restrictions. These assumptions make our problem feasible and hold in almost all real systems.

- The physical database design of each data source is known to the global mediator.
- The number of available buffer size of a data source is fixed during the entire query execution.
- Queries are answered based on up-to-date knowledge.
• Join operator is studied in this thesis; other relational operators, like aggregation and group-by, are currently not studied.

• When estimate resources consumed in a data source by a query, we do not consider caching and caching-related operations currently, but our proposed algorithm can be used together with caching and other optimization technologies.
Chapter 4

Typical System Architecture and Data Structure

In this chapter, we firstly introduce the typical architecture of database federation. We will then describe data structures needed to support the architecture. By describing the architecture and data structures, the typical execution of query optimization in database federation is also presented. We will later compare them against our proposed system architecture described in Chapter 5.

4.1 Typical Architecture of Database Federation System

Firstly, we observe the main components in typical database federations similar to those used in [3, 8, 12, 15, 20]. The system architecture is illustrated in Figure 4.
For query optimization purpose, the most relevant parts of the system are the *query optimizer* in the middleware layer (in most cases, it is the mediator), and the *wrappers* at either middleware layer or local execution layer. As in a centralized database system, the query optimizer could use a variety of different optimization algorithms, but the federated nature of the system requires that the cost estimation be made by co-working with wrappers of underlying data sources. The optimizer and the wrappers communicate through use of two constructs: (1) *Local Request* that uses the wrappers that are stored and maintained at middleware layer (in Figure 4, the Wrapper for DBMS A). Wrapper will be notified and updated once any change of semantic or catalog from the underlying data source happens. So the mediator can simply access information from the wrapper as local call. (2) *Remote Request* uses wrappers that are located in local execution layer (in Figure 4, the Wrapper for DBMS B). Every time the optimizer needs to remote call the wrapper for fresh information of data sources. In this case, relatively static information may be cached in the middleware layer. Updating the wrapper is automatically synchronized with the data source.

### 4.2 Typical Data Structures and Statistics
In this section, we describe typical database structures and statistic information used in the architecture introduced in the above section. In typical database federation, the process of query optimization utilizes the data structures and the architecture described before.

### 4.2.1 Query Execution Plan

In a typical centralized database, a query execution plan (which we call a *centralized plan*) is a binary tree consisting of the basic relational operation nodes. Basic operations usually include sequential scan, index scan, nestloop join, mergesort join, and hashjoin. Regarding to the architecture introduced in last section, centralized plans are usually produced by query compiler. Figure 5 gives an example of a centralized plan for a four way join query, called Query 4.1.

**Query 4.1:**

SELECT * FROM Relation A, Relation B, Relation C, Relation D

We call a query execution plan that specifies a distributed processing *distributed plan*. Obviously, each distributed plan is a distribution, including site selection for each operation node and data transfer assignment among data sources, of centralized plan. Figure 6 shows a possible distributed plan extent from the centralized plan in Figure 5, with the system setting that Relation A, B, C, D are located in Site 1, 2, 3, 4 respectively.

Figure 5 Centralized plan for Query 4.1
4.2.2 Statistics

To complete cost-based optimization, the optimizer and some of the wrappers need statistics as input to their cost formula. In a federated environment, it is the wrapper’s task to gather these statistics. A traditional optimizer’s collection includes probabilistic statistics (such as the number of distinct values in a column and selectivity), as well as physical characteristics (such as the number of pages it occupies and relation schema), which are used to estimate the I/O required to read the collection. In a federated environment, the optimizer still needs those statistics. In addition, hardware information such as network traffic and run-time condition of data sources might also be gathered in certain scenarios.
Chapter 5

Cluster-and-Conquer Algorithm

In this chapter, I describe Cluster-and-Conquer algorithm to perform cost-based query optimization for distributed joins over database federation with consideration of run-time conditions. The key idea of Cluster-and-Conquer is to divide query plans based on clusters, and each cluster has its optimizer to process the sub-plan with run-time conditions of data sources as consideration. This algorithm accords with many real-world observations, and it takes run-time condition into cost-based optimization with little overhead, while also simplifying the process of selecting a distributed plan.

5.1 Motivation

As mentioned in the introduction, run-time conditions should be a concern in distributed operations over database federation, because run-time conditions can significantly affect the execution cost of a query plan. However, to the best of our knowledge, there is no efficient and easy-to-implement approach that takes the run-time conditions into the optimizer’s account so far.

The difficulties of considering run-time conditions in the distributed data integration system are two-fold: (1) precise run-time condition values are not available to remote mediator, because of the transfer delay and the rapid fluctuation of the values; (2) taking run-time conditions into account will increase the complexity of optimization process.
Thus, existing systems will not consider run-time conditions, or only consider them in some limited ways (examples are the two-phase algorithm and parametric optimization introduced in Section 1.4).

To overcome the first difficulty, one optional solution is to do adaptive techniques. Once little is known in advance about the source’s properties, the system can start running with a primitive query plan, and then adjust query processing based on information gathered during the execution. The adjusting needs plan migration, or data migration or repartition, which is well known to have a large overhead [11].

When it comes to the second difficulty, an intuitive solution is using “divide and conquer” methodology, which is often used in distributed system. If the cost-based optimization and run-time conditions consideration could be done by data sources autonomously, the complexity would be divided and conquered. However, it is not always a feasible or efficient way to do so, since a total decoupled query processing cannot ensure global efficiency. Moreover, in a large scale distributed system, cooperation and inter-dependency across data sources always need to be considered, but no individual data source can do that.

In summary, to the best of our knowledge, current solutions might not be adequate for this problem so far. Our goal is to invent an easy-to-implement and efficient query optimization algorithm that is able to consider run-time conditions for distributed database federation.

### 5.2 Key Observation

From the discussion in Section 5.1, we can infer that, for an efficient query optimization over database federation, it is significant to find the tradeoff between centralized processing and distributed processing. In other words, we still want to do divide-and-conquer, but do not want to divide the system into individual, independent data sources.
To look for the right granularity of division, we investigated the following real-world facts:

(1) Many national-scale or global-scale data federations are built on the networks which consist of both broad, LAN paths and narrow, long-haul paths, as shown in Figure 7. Hence, a bunch of sites connected via LAN can communicate with each other in a fast and cheap way; while communication via WAN, or long-haul paths is slow and expensive. In this case, obviously run-time conditions can be available only within a LAN, since network delay would be insignificant and the run-time conditions of a data source can be delivered to its local “neighbor” data sources in time.

![Figure 7 A global-scale database federation overview](image)

(2) Many highly-integrated systems have to access data through a great deal of databases that belong to multiple different organizations, as shown in Figure 8. Commonly, machines of the same organization communicate via VPN (Virtual Private Network), which assures fast and cheap communication even across very large geological areas. To the contrast, accessing data located in a different organization’s site would be costly, because of necessary security check, certain access constraints or monetary charges. In such a case, it is reasonable to assume that run-time conditions are only available within an organization’s network.
After observing these situations, let us go back to our question: how to find the right granularity of division in a distributed system. We can consider all data resources in the database federation as a set of several clusters of sites, where “cluster” represents either data sources within LAN or within an organization. Figure 9 depicts this idea. Moreover, we can construct multiple levels of mediators. To simplify our following discussion, we assume that there are two layers of mediators in the database federation system: Global Mediator and Cluster Mediator. (But remember that we can define more than two layers of mediators whenever it is necessary). Global mediator and cluster mediator will cooperate to complete query processing, as shown in Figure 10.

In the next section, we will describe how these rough ideas can be implemented and verified.
5.3 Cluster-and-Conquer Algorithm

In this section, I provide the overview and details of Cluster-and-Conquer algorithm. We will firstly describe the system architecture, and then define the key data structures, and finally specify the query optimization process.

5.3.1 Proposed System Architecture
As illustrated in Section 5.2, it would be beneficial to divide the whole federation into clusters. To support this idea, we propose multi-level-mediator architecture. As shown in Figure 11, besides the global mediator, which is similar to the middleware layer in Figure 4, there are also cluster mediators located in each cluster. Both the concepts of global mediator and cluster mediator are logical. The physical implementation of the mediators has two options: (1) the mediator component is fixed in one machine, i.e. only one machine has the global mediator component and in each cluster only one machine has the cluster mediator component; (2) The mediator components are installed in multiple machines. In this case, only one global mediator and one cluster mediator in each cluster are working in each run of query processing, but different runs can use different global mediator or different cluster mediators.

The most significant parts of the system for the cluster-and-conquer approach are the query optimizers. Optimizer is the key component in each mediator, thus we have two kinds of optimizers with different functionalities based on the mediator they belong to. The query optimizer in the global mediator (we name it as global optimizer in following content) uses a System R style algorithm, which is extended to also search through the space of bushy plans. Global optimizer performs at compiling time and considers all the
tables as being stored in the clustered fashion, rather than considering tables stored in individual data source. The “clustered view” of tables is shown in Figure 12. Global optimizer’s responsibility is to determine operations to be performed across clusters and to assign sub-plans to corresponding cluster mediators. Global optimizer does not care about run-time conditions of each data source, neither does it determine operation within each cluster.

![Figure 12 Clustered View of Tables Used by Global Optimizer](image)

The query optimizer in cluster mediator (named cluster optimizer accordingly) takes a sub-plan, aka. plan fragment, as input, which is assigned by the global optimizer. Cluster optimizer is the one who actually considers run-time parameters in data sources. Because each cluster mediator is a neighbor to those data sources within that cluster, it can fetch fresh run-time conditions economically. Based on run-time information as well as static physical design, the cluster mediator can find an intra-cluster optimal plan. Note that each cluster mediator functions independently and potentially in parallel.

### 5.3.2 Proposed Data Structures and Optimization Process

We now introduce the proposed data structures used in the system. Also we describe how the Cluster-and-Conquer approach works in detail.

*Clustered plans* are produced by the global optimizer. All that the global mediator needs to do is pushing the sub-plans down to cluster mediators, coordinating communications across cluster mediators and determining inter-cluster operations. There are two types of clustered plans: (1) *Static clustered plan*, which is shown in Figure 13 (this plan tree is
based on the clustered view presented in Figure 12). The global optimizer determines this plan by referencing to physical design of tables in the federation system, including table storage information and table schemas. Static clustered plan simply assigns sub-plans to cluster mediators, without any operation determined. Cluster mediators will compute their own optimal query plans within every cluster, and return two parameters to the global optimizer: final operation site (the site where the final operation in this cluster occurs) and estimated result size. Once the global optimizer receives those parameters, it will determine the inter-cluster operations. (2) Distributed clustered plan is produced by the global optimizer to distribute a static plan. The global mediator takes all intermediate results estimated and returned by every cluster mediator into account, and select the sites for inter-cluster operations with minimum data transfer costs. In Figure 14 a distributed clustered plan is shown, where each node presented in Figure 13 has detailed information of where and how to perform the operation.

As we mentioned above, the cluster optimizer is responsible to find the optimal execution plan in its cluster. Cluster mediator takes a list of tables to be joined as input, which is
assigned by the global mediator, and then collects run-time conditions and statistics from data sources. By employing a customized cost model, which is discussed in Section 3.1, the cluster mediator produces an optimal physical execution plan for all the joins in the cluster. An example of the execution plan is shown in Figure 15, which is produced by the cluster mediator of Cluster 1 for the left sub-tree in Figure 13. Here each node represents a physical operator, and the location where the operator is performed is also explicated. So the operator tree explicates the flow of data transfer as well.

![Figure 15 Physical Execution Plan for Cluster 1](image)

Having the cluster mediator handle the intra-cluster scheduling autonomously has three-fold benefits. Firstly, the communication within a cluster is time-efficient, so the value of run-time parameters collected by the cluster mediator is much fresher than that gathered by the global mediator. Secondly, each cluster mediator can deal with its own query concurrently, which implicatively employs the independent parallelism. Thirdly, the complexity of the centralized optimization of a whole query plan in distributed environment is greatly decreased, since cluster mediators can conquer every piece of less complex sub-plan respectively.

### 5.3.3 Algorithms and Running Example

In this section, we provide pseudo-code for the algorithm employed by the global mediator and the one implemented by the cluster mediator. Algorithm 1 gives the pseudo-code for global optimization. Algorithm 2 gives the pseudo-code for cluster optimization.

**Algorithm 1:** GlobalOptimization (list of table names tables, clusteredView)
// Firstly construct static clustered plans

define hash table tablesInCluster(clusteredName, listTables)
for each table name tableName in tables
    get clusterName of tableName from clusterViews
    if clusterName exists in keys of tablesInClusters
        add tableName to listTables on this clusterName
    else
        add tableName to new list of table names tmpListTables
        add pair (clusterName, tmpListTables) to tablesInCluster
    end if
end for

// after constructing static clustered plans, send to corresponding cluster mediators
for each entry (clusterName, listTables) in tablesInCluster
    send listTables to the cluster mediator with name clusterName
end for

// after sending sub-plans, wait for returned parameters
loop
    wait for returned parameter set (finalSite, resultSize) from each cluster mediator
end loop

//after gathering parameters needed, compute inter-cluster operations
for all parameter sets
    determine each inter-cluster operation as (clusterName1, clusterName2, siteName)
    using greedy on minimization of intermediate result size // given by resultSize
    send clusters with name clusterName1, clusterName2 the operation
end for
In Cluster-and-Conquer algorithm, the global mediator is responsible for the “to cluster” part. The input for global algorithm is a list of tables that need to be joined, as well as a clustered view of the whole federation, as shown in the Algorithm 1. There are two kinds of inputs in different running phases. The first is a static clustered plan, and the second is a distributed clustered plan. Algorithm 1 communicates with certain cluster optimizers over network.

Algorithm 2: ClusterOptimization (list of table names *tables*)

```java
// to find smallest intermediate result
define structure interMediateSize as (numTuples, tableL, tableR)
define interSize, minSize as instance of interMediateSize

for every two tables tableL, tableR
    interSize = estimateJoinSize(catalog of tableL, catalog of tableR)
    if interSize.numTuples < minSize.numTuples
        minSize = interSize
    end if
end for

// heuristically build execution plan
define operation tree ePlan
while (table in tables is not added to ePlan)
    for the rest of tables
        currentTable = smallestInter(ePlan, rest of tables)
        add currentTable to ePlan
    end for
end while

// cost-based select operation site
for each join node joinNode in ePlan
    // the comparison is based on the predefined cost model
    if runtimeCondition (joinNode.leftChild) > runtimeCondition (joinNode.rightChild)
        joinNode.opSite = join.leftChild.opSite
    else
        joinNode.opSite = join.rightChild.opSite
    end if
```
Accordingly, in Cluster-and-Conquer algorithm, the cluster mediators are responsible for the “to conquer” part. For each execution, each cluster optimizer that receives sub-plan from the global mediator will run Algorithm 2. The input sub-plan for the cluster optimizer is in fact a list of table(s) located in that cluster. The output of Algorithm 2 is an optimal physical execution plan, and the optimizer will dispatch this execution plan to corresponding data sources.

I will use Query 5.1 as a running example to demonstrate the algorithms. Notice: Query 5.1 and Figure 16 are taken from TPC Benchmark H Standard Specification Revision 2.8.0.

Query 5.1:
select *
from PART, PARTSUPP, SUPPLIER, NATION, CUSTOMER
where PART.PARTKEY = PARTSUPP.PARTKEY
and PARTSUPP.SUPPKEY = SUPPLIER.SUPPKEY
and SUPPLIER.NATIONKEY = NATION.NATIONKEY
and NATION.NATIONKEY = CUSTOMER.NATIONKEY
The physical design of these tables is presented in Figure 16. The legend of Figure 16 is: the arrows point in the direction of the one-to-many relationships between tables; the number below each table name represents the cardinality of the table. SF stands for Scale Factor, to obtain the chosen database size.

![Table Schemas for Query 5.1](image)

Figure 16. Table Schemas for Query 5.1

The physical setting used in executing Query 5.1 by the algorithm is illustrated in Figure 17, where we have two clusters, each cluster contains several data sources and those data sources who store tables used in the query are specified. Notice that the clusters are determined by either network environment in terms of LAN/WAN, or enterprise boundaries in terms of VPN/Internet.
Given the above setting and incoming query, the global optimizer that runs $Algorithm1$: $GlobalOptimization$ will firstly produces a static clustered plan, as shown in Figure 18.

The sub-plan sent to cluster mediator of Cluster1 is the left sub-tree, while that for Cluster2 is the right sub-tree according to Figure 18. Cluster optimizers in Cluster1 and Cluster2 will then run $Algorithm2$: $ClusterOptimization$. Cluster1’s optimizer generates its own optimal execution plan as shown in Figure 19 (a) and Cluster2’s optimal execution plan is shown in Figure 19 (b).
Cluster optimizers return their final operation nodes and estimated result sizes to the global mediator. At the same time, the cluster executors will start execution of the query based on their own optimal plans.

Global optimizer uses the returned information and runs the second-half part of Algorithm 1 (line 21-26). Assume that the intermediate result of Cluster1 has smaller size than that of Cluster2, so the global optimizer determines that the inter-cluster join will be done at \( m22 \). The optimizer will send this command to both cluster mediators and then the inter-cluster join will be done accordingly. Finally the join result is output.

### 5.4 Cost Model Analysis

The cost model used for intra-cluster operations by each cluster optimizer is Formula 1.1, and the cost model used for the whole query plan by the global mediator is Formula 1.3, both of which are defined in Chapter 3. To estimate the cost value of an execution plan, all the fields will be scaled to millisecond. We use Query 5.1 as an example to show how this cost model used in the prototype system to find the optimal plan.

Given a typical environment setup (which we will introduce in detail in Chapter 6), from the execution plans generated by cluster mediators as shown in Figure 19, global mediator will produce the whole plan as depicted in the following figure. Plan (b) and (c) are two equivalent plans selected by other algorithms. Notice that in Plan (a), the operation on \( m13 \) and the operation on \( m22 \) can be executed in parallel. So for Plan (a), the cost is calculated with Formula 1.3. While in Plan (b) and Plan (c), we cannot execute any operations in parallel, so their costs are calculated with Formula 1.1. Based on the cost models, we scale the cost of Plan (a) as 1, then the cost of Plan (b) is 1.28 and the cost of Plan (c) is 1.21. This demonstrates that our algorithm does find an optimal plan with the defined cost model. Our experimental result presented in Chapter 6 also verified this conclusion.
5.5 Algorithm Design Analysis

In this section, we analyze our Cluster-and-Conquer algorithm in the terms of query plan search space and run-time condition consideration and overhead.

5.5.1 Query Plan Search Space

The philosophy of cost-based optimization is to search a space of equivalent query plans, and to find the optimal plan with minimum cost. In distributed data management system, besides searching for algebraic space, the optimizer need to also search site-selection space. Thus the number of candidate plans could be huge, which means exhaustive search is impractical. Commonly some heuristic optimization process will be performed to narrow down the search space before the cost-based process. Cluster-and-Conquer algorithm performs the heuristic process by global optimizer (as listed in Algorithm 1 line 1-12). The heuristic rule to filter query plans is based on the clustered view of the federation: joins within a cluster will be executed first and then inter-cluster joins are executed afterwards.

Before reasoning our heuristic rule, let us study a typical way of heuristic query selection in distributed database called *maximum-push-down*. This approach pushes all join operations down to data sources, and tables located in a same data source will be joined first. This approach has been proved to be inefficient in many cases [8] because it does not consider cost (in terms of disk I/O, data transfer delay, etc.) at all. However,
maximum-push-down decreases the plan search space tremendously. Our heuristic rule has a flavor of maximum-push-down, but the key difference is that we push joins down to *clusters*, rather than single data source. And the optimization algorithm (*Algorithm2*) performed by cluster optimizers is cost-based.

Considering that only a subset of plans will be fully explored in cost-based optimization, we may expect this approach to produce much worse plans than exhaustive algorithm. However, remember that the clustering is based on the properties of a database federation (which are addressed in Section 5.2), such as: there exist enterprise boundaries, which forbid moving data to other enterprises’ sites; or in a global database federation, data transfer through long-haul paths is pretty costly, while data transfer within a LAN is economical. Hence joining primitive tables across distinct clusters is either infeasible or prohibitive, which suggests performing cost-based optimization over those plans is unnecessary.

### 5.5.2 Run-time Condition Consideration and Overhead

Cluster optimizers are responsible for collecting and considering run-time conditions of data sources. There are mainly two efforts: (1) communicate with every data source in the given query; (2) calculate the cost for each candidate plan with the cost model which includes the run-time condition parameter(s). The first effort will increase the optimization process time due to the communication cost and the second effort will make the cost calculation slightly more complex. Those overheads will be studied further by running experiments and comparing with other related optimization approaches in Chapter 6.
Chapter 6

Experimental Evaluation

In this section, I describe experimental results evaluating the performance of Cluster-and-Conquer approach compared to the maximum-push-down approach and the two-phase approach of processing distributed joins.

6.1 Experimental Setup

6.1.1 Prototype System Implementation

We have implemented a distributed data federation system to test out our hypothesis. The system is implemented using Java, and it is capable of optimizing and executing distributed join queries across a set of machines connected by network. The architecture of the system is depicted in Figure 11. The system is deployed on three machines. The basic simulation principles are: (1) using one machine to simulate a cluster of machines; (2) setting network factors to simulate data transfers over LAN and WAN, as well as VPN and Internet; (3) in each machine using multiple isolating file spaces to simulate different data sources.

In each simulated data source, there are many source tables stored. Table storage has no overlap among data sources, which accords with typical data integration system
environment. There is only one global mediator who serves as application interface sitting on one machine. The global mediator accepts input queries and will return final query results. Within each machine, there is one cluster mediator program.

Query execution module, which is the database query engine in fact, is implemented by myself, because I want to have more flexibility to set operation parameters. One machine has one copy of execution module, but the program is written in a multi-threaded manner. As we mentioned before, one machine contains several simulated data sources, and we assume that every data source is capable to execute basic database queries, so the execution model will fork execution threads for each data source once the system starts running. The basic setup is presented in Figure 20. The CDC machine where the global mediator is located can be also considered as a cluster, once we store tables and install execution module on that.

Figure 20 Basic Experiment Architecture Setup

For join method, we implemented sort-merge join and the sort function is implemented as *multi-way external sorting*. The reason for the join method implementation is two-fold: sort merge join is very memory sensitive which is ideal for our experiment goal; for large size table we need external join, and external sorting is both memory sensitive and
efficient. This did not affect the performance comparisons among optimization algorithms discussed below.

6.1.2 Data Sets and Queries

We use TPC-H benchmark data sets as source tables. TPC-H queries are modified by removing other operations than joins, in order to be executable in our prototype system. We have tested the following three queries:

Query 6.1 and Environment setup.
Size of input queries: five tables. Number of clusters: two. Table storage: as shown in Figure 17. SQL for the query:

```sql
select *
from PART, PARTSUPP, SUPPLIER, NATION, CUSTOMER
where PART.PARTKEY = PARTSUPP.PARTKEY
and PARTSUPP.SUPPKEY = SUPPLIER.SUPPKEY
and SUPPLIER.NATIONKEY = NATION.NATIONKEY
and NATION.NATIONKEY = CUSTOMER.NATIONKEY
```

Query 6.2 and Environment setup.
Size of input queries: four tables. Number of clusters: one. Table storage: all tables are in the same cluster. SQL for the query:

```sql
select *
from CUSTOMER, ORDERS, LINEITEM, PARTSUPP
where CUSTOMER.CUSTKEY = ORDERS.CUSTKEY and
ORDERS.ORDERKEY = LINEITEM.ORDERKEY and
LINEITEM.PARTKEY = PARTSUPP.PARTKEY and
LINEITEM.SUPPKEY = PARTSUPP.SUPPKEY
```

Query 6.3 and Environment setup
Size of input queries: six tables. Number of clusters: three. Table storage: NATION, CUSTOMER are in Cluster 1; REGION, ORDERS are in Cluster 2; LINEITEM, PARTSUPP are in Cluster 3. SQL for the query:

```sql
select *
```
from REGION, NATION, CUSTOMER, ORDERS, LINEITEM, PARTSUPP
where REGION.REGIONKEY = NATION.REGIONKEY and
    NATION.NATIONKEY = CUSTOMER.NATIONKEY and
    CUSTOMER.CUSTKEY = ORDERS.CUSTKEY and
    ORDERS.ORDERKEY = LINEITEM.ORDERKEY and
    LINEITEM.PARTKEY = PARTSUPP.PARTKEY and
    LINEITEM.SUPPKEY = PARTSUPP.SUPPKEY

6.1.3 Network Simulation

The network will be simulated using the message cost model introduced in [12]: A data set of size n bytes takes

$$\alpha + \beta \times n \quad (\alpha \text{ is the start-up cost and } \beta \text{ is the transfer cost per Mb})$$

to reach the other end. In our experiments, the cost is scaled into millisecond. By setting the cost parameters we can simulate local area network as well as wide area network.

6.2 Overview of Comparable Approaches

We conducted experiments to compare the proposed Cluster-and-Conquer algorithm with other two well-known related works, which we call Maximum-push-down approach and Two-phase approach respectively in the following contents.

Maximum-push-down approach is pure heuristic. It has been suggested [32, 33] that pushing as much work as possible to the data sources is sufficient. For those operations across data sources, this approach allows arbitrary selection of the data source which will execute the operation. This implementation references to the “Pushdown join execution” described in [8].

Two-phase approach was originally invented for parallel systems [15]. We implemented this approach in a simplified manner: An optimizer takes tables to join as input and calculates an logical optimal plan using cost-based algorithm based on minimizing intermediate result size and estimated I/O cost, without considering any network or
system condition; A distributor takes the logical optimal plan as input and assigns data sources to execute operations based on run-time conditions gathered from data sources. Both the optimizer and the distributor are centralized, and set on one mediator-like machine.

6.3 Varying Available Buffers

In this section, we evaluate Cluster-and-Conquer algorithm in terms of varying available buffers in every data source. The performance of Cluster-and-Conquer algorithm is compared with Maximum-push-down approach and Two-phase algorithm.

We design five scenarios according to real-world system environment as follows. The detailed settings of these scenarios are listed in Table 2. The exact value of the run-time available buffer is a random number within the specified range in each grid.

- Scenario 1 is to simulate the environment that every data source has uniform run-time conditions.
- Scenario 2 is to simulate the environment that run-time conditions of data sources are nearly even, and data sources which hold large-size tables, i.e. PARTSUPP, PART, have relatively larger available buffers.
- Scenario 3 is to simulate the environment that run-time conditions of data sources are random and independent of any data size.
- Scenario 4 is to simulate the environment that run-time conditions of data sources are non-monotonic to the data size, which means data sources which hold large-size tables, i.e. PARTSUPP, PART, have relatively smaller available buffers. Also the run-time conditions are not even.
- Scenario 5 is to simulate the environment that run-time conditions of data sources are largely different. Intuitively, if the system randomly chooses data sources to perform operations, the performance could be pretty bad.

In this group of experiment, network factors are stable. For communication within cluster, we set $\alpha = 10$, $\beta = 0.05$; For communication across clusters, we set $\alpha = 20$, $\beta = 0.2$. 

42
This setting is based on the general network bandwidth/throughput difference between LAN and WAN [3, 34].

<table>
<thead>
<tr>
<th>Available Buffer (MB)</th>
<th>Scenario 1</th>
<th>Scenario 2</th>
<th>Scenario 3</th>
<th>Scenario 4</th>
<th>Scenario 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>m11(PART)</td>
<td>128-256</td>
<td>64-128</td>
<td>256-512</td>
<td>32-64</td>
<td>128-256</td>
</tr>
<tr>
<td>m12(PARTSUPP)</td>
<td>128-256</td>
<td>128-256</td>
<td>64-128</td>
<td>64-128</td>
<td>32-64</td>
</tr>
<tr>
<td>m13(SUPPLIER)</td>
<td>128-256</td>
<td>128-256</td>
<td>32-64</td>
<td>32-64</td>
<td>256-512</td>
</tr>
<tr>
<td>m21(NATION)</td>
<td>128-256</td>
<td>64-128</td>
<td>128-256</td>
<td>256-512</td>
<td>64-128</td>
</tr>
<tr>
<td>m22(CUSTOMER)</td>
<td>128-256</td>
<td>64-128</td>
<td>128-256</td>
<td>128-256</td>
<td>512-1024</td>
</tr>
</tbody>
</table>

Table 2: Scenario Setting with Varying Buffer Size in Each Data Source.

Query 6.1, 6.2, and 6.3 were executed in every scenario with three approaches respectively (“pushdown” denotes the Maximum-push-down approach, “2phase” denotes Two-phase approach, and “c&c” denotes Cluster-and-Conquer Algorithm) for five times, and the average running time are recorded.

Figure 21 shows the result of average execution time of running Query 6.1 in five scenarios with three approaches respectively. The experimental results reflect the necessity to consider run-time conditions for optimization, as well as shows Cluster-and-
Conquer algorithm is way better than Two-phase algorithm. In Scenario 1, because considering run-time condition (here we mean available buffer size) does not benefit, Max-push-down approach performs not bad. However, when the available buffers gradually dominate the cost of query processing, not considering this condition incurs drastic defect. When it comes to the comparison between Two-phase and Cluster-and-Conquer, the results show that when the impact of network communication is comparable to the impact of available buffer size (Scenario 2 and 3), Cluster-and-Conquer produces more efficient result.

Figure 22: Running Time of Query 6.2 with Varying Buffer Size

Figure 23: Running Time of Query 6.3 with Varying Buffer Size
Figure 22 and Figure 23 depict the results of running Query 6.2 and Query 6.3. These results also reflect the benefit of considering run-time condition. However, in the results of running Query 6.2, Cluster-and-Conquer algorithm does not beat Two-phase algorithm considerably, while in that of Query 6.3, Cluster-and-Conquer prevails. To understand this phenomenon, we review the main distinctions between them: (1) Two-phase does not take the network condition into account, i.e. its cost model only includes resources consumed and available buffer factors. Thus when network-related cost is nontrivial, Two-phase may select non-optimal plans. (2) As we assume, the value of available buffer size is returned by every data source once it is inquired. Two-phase has only one global mediator who collects these values from every data source, so the cost of messaging will be considerable especially when inter-cluster communication is expensive. Cluster-and-Conquer has each cluster mediator to handle local available memory information, so the inter-cluster messaging merely happens at most as many as the number of clusters (which is usually much less than the number of data sources).

### 6.4 Varying Network Factors

In this section, we evaluate Cluster-and-Conquer algorithm in terms of varying network factors, i.e. the difference of network conditions between inter-cluster communication and intra-cluster communication. The performance of Cluster-and-Conquer algorithm is also compared with Maximum-push-down approach and Two-phase algorithm.

As a reminder, the cost of network transfer is simulated as $\alpha + \beta \ast n$ in terms of millisecond. The scenarios are varying with the trend that the throughput difference between inter-cluster and intra-cluster is increasing.

<table>
<thead>
<tr>
<th>Scenario 1</th>
<th>Scenario 2</th>
<th>Scenario 3</th>
<th>Scenario 4</th>
<th>Scenario 5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LAN/VPN</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\alpha = 10$, &amp; $\beta = 0.1$</td>
<td>$\alpha = 10$, &amp; $\beta = 0.1$</td>
<td>$\alpha = 10$, &amp; $\beta = 0.05$</td>
<td>$\alpha = 5$, &amp; $\beta = 0.02$</td>
<td>$\alpha = 2$, &amp; $\beta = 0.01$</td>
</tr>
<tr>
<td><strong>WAN/Internet</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\alpha = 10$, &amp; $\beta = 0.1$</td>
<td>$\alpha = 20$, &amp; $\beta = 0.15$</td>
<td>$\alpha = 30$, &amp; $\beta = 0.2$</td>
<td>$\alpha = 35$, &amp; $\beta = 0.3$</td>
<td>$\alpha = 40$, &amp; $\beta = 0.4$</td>
</tr>
</tbody>
</table>

Table 3 Scenario Setting with Varying Network Factors
Query 6.1 is executed in every scenario with three approaches respectively. The resulting time is the average of several executions. In this group of experiments, the available buffer size of each data source is stable. The setup of available buffer size is the same as Scenario 4 in Table 2.

Figure 24 depicts the result of running Query 6.1 with varying network factors. We can see that Cluster-and-Conquer algorithm performs more efficiently than other two approaches. We observed that Max-push-down always executed the same query plan given the input query, nevertheless the run-time conditions changed. Two-phase did not do much better than Max-push-down, because it only considers available buffer size, which keeps still in this group of experiments and then it also sticks to one execution plan. From our observation, the execution plan chosen by Two-phase approach contains at least one unnecessary inter-cluster data transfer operation, which required significant time when WAN/Internet factors are high. To the contrast, Cluster-and-Conquer algorithm restricts the amount of inter-cluster communications by dividing the whole query plan into cluster-based sub-plans. Thus, as the speed difference between LAN and WAN increases gradually, Cluster-and-Conquer algorithm demonstrates more advantages.

Figure 24 Running Time of Query 6.1 with Varying Network Factors

6.5 Cost of Optimization
In this section we study the running time of optimization algorithm itself, which is also known as *cost of costing* [12]. In this group of experiments, we keep the available buffer size (using Scenario 3 in Table 2) of each data source still, and the network factors are changed for three scenarios. By varying the input query size, in terms of number of tables to join, we monitored the running time of optimization process. This time is recorded from when the optimization module is called to when the module finishes with an “optimal” plan as output. We made up some input queries with significant number of tables, but we stopped the processing once optimization was done (without actually executing the plans).

Figure 25 gives the cost (in terms of running time) of the three optimization algorithms. There are three sizes of input query, and three network conditions. “Expensive” denotes the Scenario 5 in Table 3, “Middle” denotes Scenario 3 in Table 3, and “Cheap” denotes Scenario 2 in Table 3. We can see that Max-push-down approach has minimum and relatively stable cost, simply because it does not implement cost-based optimization.
Two-phase approach is very sensitive to network conditions, as we explained before, because of its considerate messaging cost. Two-phase approach is also affected obviously by size of input query. This is because it adopts centralized plan enumeration and selection, and the increase of the number of tables will significantly complicate the algorithm. So far, Cluster-and-Conquer has acceptable cost, in the sense that it can adapt to the network condition well and it has distributed plan enumeration and selection process, which can transform a big query plan into small pieces.
Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis firstly focuses on general query optimization technologies over database federation systems. Given that database federation is by nature a distributed system, as well as data integration system, most query optimization approaches designed for distributed databases and data integration systems are also adopted in database federations. I studied many related work and found out that very few works addressed the problem of considering run-time conditions in query optimization. By analyzing both theoretically and experimentally, I present the need to take run-time conditions, including the available buffers and CPU utilities in the data sources and network environment, into account in optimization processing. Also I pointed out the challenges of doing this consideration.

Secondly this thesis studies two existing approaches, namely parametric algorithm and two-phase algorithm, which are potentially able to consider run-time conditions in the optimization process of database federations. However, after analyzing their pros and cons, we found that both of them are not sufficient for optimization of distributed joins in database federations.

Thirdly, given our target optimization approach is cost-based and is used in distributed environment, cost model definition and parallelism constraints are presented. And then typical database federation system architecture and data structures are introduced.
Fourthly, I proposed Cluster-and-Conquer algorithm for optimizing distributed join over database federation with efficiently considering run-time conditions. Cluster-and-Conquer algorithm is motivated from real-world observation as well as the defects of existing system architecture. Since run-time conditions of data sources are prone to fluctuate, only closely connected “neighbor” machines are able to get fresh information. And real-world public network and enterprise network environment suggests an intuitive way to determine “closely connected” machines based on network data transfer cost. So we proposed to view the whole database federation as clustered system, and provide each cluster of data sources with its cluster mediator. Based on this architecture, the query optimization can be divided into two procedures: the global optimizer decides inter-cluster operations, and cluster optimizers handle the sub queries that happen in those data sources within the cluster with run-time condition consideration. Surprisingly, besides being able to deal with run-time conditions, Cluster-and-Conquer algorithm also outperforms other existing works in terms of “cost of costing”. This is mainly because unnecessary inter-cluster operations are naturally removed, and also each cluster optimizer only needs to process a sub query plan which is much simpler than dealing with a whole distributed query plan by one centralized optimizer, moreover network messaging cost is decreased.

Finally we implemented the prototype federation system with the proposed architecture and optimization algorithm. The experimental results showed the capabilities and efficiency of Cluster-and-Conquer algorithm and gave the target environment where the algorithm performs better than other related approaches.

### 7.2 Future Work

For future work, we plan to extend our study in the following directions.
The Cluster-and-Conquer algorithm assumes the clustered view of data sources is given as input. So a natural extension is to enable the algorithm to gather this information by itself. Currently the prototype system has two levels of mediators, but it is necessary to extend the system in order to support multi-level mediators whenever the environment demands.

Another possible extension is to employ this algorithm to other distributed systems, such as distributed databases and grid computing systems. The philosophy of cluster-and-conquer is expected to be useful for large-scale distributed computing environments.

We may also plan to extend this algorithm for the processing of other types of operations, like aggregate (such as group-by, max and min), top-K, etc. The thesis mainly discusses distributed join operation. Certainly we can do join firstly and then perform other operations on the joined result, but there can be other brilliant way to schedule all operations efficiently in distributed environments.
References


