Doctoral Dissertation Academic Year 2023

A Data-Driven System for Combinatorial Optimization Problems

Graduate School of Media and Governance

Keio University

Kazuhiro Saito

Abstract of Doctoral Dissertation

In recent years, data-driven systems such as machine learning are expected to solve social problems. the data-driven systems need to connect to various databases to collect the necessary data. In addition, when the problem of utilizing the predicted results from machine learning involves combinatorial optimization, it is difficult to achieve the desired performance using existing classical optimization methods due to the large scale of the problem. To solve these issues, this thesis proposes a data-driven system for combinatorial optimization problems with a virtual integrated data platform by data virtualization and a high-performance optimization environment by quantum annealing. This system integrates multiple data sources into a single virtual database system through data virtualization and provides a unified interface to users without modifying existing data sources. Furthermore, quantum annealing, a metaheuristic for combinatorial optimization based on quantum effects, enables fast solution of combinatorial optimization problems that are difficult to solve with conventional classical computers. We have implemented the proposed data-driven system for several data sources and used an actual quantum annealing machine, in order to evaluate the impact of communication between the data virtualization system and data sources on the system performance and to quantitatively compare quantum annealing and classical methods. The evaluation results show that the proposed data-driven system is effective for large-scale data in data infrastructure environments where a single query processing engine shares multiple data sources, as well as when a single data source is partitioned into multiple data sources. We also show that quantum annealing is effective in the vehicle routing problem and the grouping problem as specific combinatorial optimization problems, and discuss the overall performance of the proposed data-driven system from data collection to solving these optimization problems.

Keyword: Combinatorial Optimization Problem, Data Platform, Data Virtualization, Quantum Annealing

論文要旨

近年，機械学習等のデータ駆動システムによる社会問題の解決が期待されている．そのようなシステムでは様々なデータベースへ接続して必要なデータを収集する必要がある．また，機械学習で予測した結果を活用する問題が組合せ最適化を含む場合に，その問題の規模が大きくなることで既存の古典的な最適化手法では所望の性能を出すことが困難となる課題もある．本論文では，これらの課題を解決するために，データ仮想化によるデータ基盤の仮想統合環境と，量子アニーリングによる高性能な最適化環境を備えた組合せ最適化問題のためのデータ駆動システムを提案する．本システムは，データ仮想化によって複数のデータソースを仮想的に一つのデータベースシステムとして統合し，既存のデータソースに変更を加えずに，ユーザに対して統一のインタフェースを提供することができる．更に、量子効果を利用した組合せ最適化のメタヒューリスティック手法である量子アニーリングによって，従来の古典コンピュータで解くことが困難であった組合せ最適化問題を高速に解くことできる．提案するデータ駆動システムのプロトタイプを実装し，実際の量子アニーリングマシンを利用して，データ仮想化システムとデータソース間の通信がシステムの性能に与える影響を評価し、量子アニーリングと古典的な手法を定量的に比較する．評価結果として，提案するデータ駆動システムが，大規模なデータを対象に，一つのクエリ処理エンジンが複数のデータソースを共有するデータ基盤環境や，一つのデータソースを複数のデータソースに分割した場合においても有効であることを示す．また，具体的な組合せ最適化問題として配送計画問題とグルーピング問題において量子アニーリングが有効であることを示し，提案するデータ駆動システムにおいてデータ収集からこれらの最適化問題を解くまでの全体の性能を議論する．

キーワード：組合せ最適化問題，データ基盤，データ仮想化，量子アニーリング

Acknowledgements

First of all, I would like to express my deepest gratitude to my supervisor, Associate Professor Hideyuki Kawashima, throughout this research. His continuous research guidance since 2014, when I started collaborative research with him as a businessman before entering this program, has laid the foundation for me as a researcher. After I started this program, he continued to support me more closely, giving me a lot of advice in my research area and providing in-depth knowledge not only on research methodology but also on fostering future researchers. I would like to express my gratitude again.

As associate reviewers of the Research Advisory Group, I would like to express my sincere gratitude to Professor Rodney Van Meter, Professor Yasushi Kiyoki, and Professor Onizuka of Osaka University for their valuable advice on the thesis proposal. I would like to thank Professor Rodney Van Meter, Professor Jin Mitsugi, Associate Professor Shu Tanaka, and Professor Onizuka of Osaka University for their valuable advice in this doctoral thesis review.

In the Data Platform Research Group, Mr. Natsuki Hamada and I have been collaborating on the use of quantum annealing since the beginning of this program. Mr. Hiroaki Uno, Ms. Risa Ochiai, and Ms. Mengfei Shen, who are graduates, have also collaborated with me on research related to the data platform. These collaborations have greatly contributed to my growth through their discussions. I would like to express my deep appreciation to the above members of the laboratory.

I would like to thank Dr. Chihiro Ono, Mr Atsunori Minamikawa, Mr. Mori Kurokawa, and Mr. Arei Kobayashi, my supervisors at KDDI Research Inc. We are grateful to them for their understanding to enter this course for working people, despite their busy schedules. In addition, they have concluded a joint research agreement between the company and Associate Professor Hideyuki Kawashima, and have given us a great deal of advice on our research in this course as part of their company activities. I would like to express my sincere gratitude for their supports for the activities of this program. I

would also like to express my sincere gratitude to my colleague, Mr. Shigeki Muramatsu, Mr. Yasuyuki Watanabe, Mr. Nobuyuki Maita, Mr. Shigeki Oyama, Mr. Tomoaki Umeki, Mr. Akio Ikami, Dr. Akihiro Kobayashi, and Dr. Yuichi Ishikawa for their great support in technical discussions and review of papers.

Finally, I would like to express my sincere gratitude to my wife, who understood the writing of this doctoral thesis in addition to her company activities and gave me tremendous support, including in my personal life.

September, 13, 2023

Kazuhiro Saito

Contents

Chapter 1 Introduction 1

1.1 Background 1

1.2 Related Paradigm 2

1.3 The Problem 4

1.4 Research Approach 5

1.5 Contributions 5

1.6 Organization of Thesis 6

Chapter 2 A Data-Driven System for Combinatorial Optimization Problems 8

2.1 System Architecture 8

2.2 Data Virtualization 9

2.2.1 Components of Data Virtualization System 10

2.2.2 Issues of Data Virtualization 10

2.3 Quantum Annealing 11

2.3.1 How to Use 11

2.3.2 Issues of Quantum Annealing 12

2.3.3 Related Technologies 13

Chapter 3 Data Virtualization for Query Engines Sharing Data 16

3.1 Introduction 16

3.1.1 Background 16

3.1.2 Contributions 17

3.1.3 Use Cases 18

3.2 The Problem 19

3.2.1 Issues of the Data Virtualization System 19
3.2.2 Related Work 20
3.3 The Proposed System 21
3.3.1 Overview 21
3.3.2 Assumptions 23
3.3.3 Virtual Schema 24
3.3.4 Query Splitting Method 25
3.3.5 System Management 32
3.4 Implementation of the Prototype System 33
3.4.1 Target systems 33
3.4.2 Selection Criteria of QE in the Prototype Systems 33
3.4.3 Implementation 35
3.5 Evaluation 36
3.5.1 System evaluation (Hive / Impala) 37
3.5.2 System evaluation (SQL-on-Hadoop / Solr) 43
3.5.3 Discussion 45
3.6 Summary 51

Chapter 4 Data Virtualization for Dividing of Large Database System 52

4.1 Introduction 53
4.1.1 Background 53
4.1.2 Contributions 54
4.2 Related work 54
4.3 Query pushdown method 57
4.4 Data placement analysis method 57
4.5 Intermediate data relocation method 58
4.6 Implementation 60
4.6.1 Query Log DB 60
4.6.2 Data Placement Analysis Tool 62
4.6.3 Data Virtualization System 63
4.7 Evaluation 63
4.7.1 Evaluation Environment 64

4.7.2 Summary of Evaluation Results 67
4.7.3 Details of Evaluation Results 68
4.7.4 Discussion 71
4.8 Summary 75

Chapter 5 Evaluation of Vehicle Routing Problem using Quantum Annealing 76

5.1 Introduction 76
5.1.1 Background 76
5.1.2 Contributions 77
5.2 Applying Quantum Annealing to VRP 78
5.2.1 Problem Definition 78
5.2.2 Mathematic Model 79
5.2.3 QUBO Model 81
5.3 Evaluation 82
5.3.1 Evaluation Environment 83
5.3.2 Evaluation Data 84
5.3.3 Performance Evaluation of Quantum Annealing 86
5.3.4 Scaling Evaluation 88
5.4 Discussion for the Use of Quantum Annealing 91
5.5 Related Work 93
5.6 Summary 94

Chapter 6 Evaluating Quantum Annealing for Grouping Optimization with all Members' Compatibility 95

6.1 Introduction 96
6.1.1 Background 96
6.1.2 Contributions 97
6.2 Grouping Optimization using Quantum Annealing 98
6.2.1 Problem Settings 98
6.2.2 Mathematical Model 98
6.2.3 Formulation for Quantum Annealing 100
6.3 Evaluation 101
6.3.1 Evaluation Settings 101

6.3.2 Tuning Penalty Coefficients 102
6.3.3 Performance Evaluation 104
6.4 Summary 105

Chapter 7 Conclusion 106
7.1 End-to-End Performance of our proposed data-driven system 106
7.1.1 VRP 107
7.1.2 Grouping Optimization 108
7.2 Future direction of this research 109
7.2.1 Data Virtualization 109
7.2.2 Quantum Annealing and Quantum Computing 110
7.2.3 Machine Learning System 112
7.3 Conclusion remarks 113

References 114

List of Figures

Figure 1-1: Traditional data-driven system architecture for combinatorial optimization problem. 3
Figure 2-1: Proposed data-driven system architecture for combinatorial optimization problem, which includes data virtualization system in data platform and quantum annealing machine for solving combinatorial optimization. 9
Figure 3-1: A use case of our proposal system. 19
Figure 3-2: Virtual schema: existing system and proposal system. 20
Figure 3-3: System architecture of proposed data virtualization system for data sharing storage. 22
Figure 3-4: System components and an example of executing a split query. 23
Figure 3-5: Processing flow of the query splitter. 25
Figure 3-6: A sample user query. 36
Figure 3-7: Sample split queries generated by the prototype system. 36
Figure 3-8: Areas of subqueries that QE executes in each TPC-H query. 38
Figure 3-9: Performance evaluation results on SQL-on-Hadoops with Scale 50. . 39
Figure 3-10: Performance evaluation results on SQL-on-Hadoops with Scale 100. 39
Figure 3-11: Performance evaluation results on SQL-on-Hadoops with Scale 150. 40
Figure 3-12: Performance evaluation results on SQL-on-Hadoops with Scale 200. 40
Figure 3-13 Evaluation Results of the overhead. 41
Figure 3-14: Estimated maximum memory sizes of each subquery on scale 150. 42
Figure 3-15: Error rates of estimated maximum memory size. 43

Figure 3-16: Evaluation query for tweet data analysis. 44
Figure 3-17: Evaluation results for performance on Solr and SQL-on-Hadoop. .. 45
Figure 3-18: The split queries for Solr and SQL-on-Hadoop, used for the evaluation in Figure 3-17. 45
Figure 3-19: Execution time of query Q4 by Impala, the prototype system and Hive, and relative performance of Impala against Hive. 46
Figure 3-20: Relative performance ratio of our prototype against Hive 47
Figure 3-21: Breakdowns of execution times on the prototype system. 49
Figure 3-22: Comparisons of error rates of the selectivity-style method and the histogram-style method. 50
Figure 4-1: Data virtualization System 55
Figure 4-2: A virtual integrated environment using data virtualization. 56
Figure 4-3: An example of the intermediate data relocation. 59
Figure 4-4: An example of a query execution plan log in Vertica. 61
Figure 4-5: Query execution plan of Figure 4-4, which is represented by Nested Set Model. 61
Figure 4-6: Tree structure of Figure 4-5 and position assignment procedure. 62
Figure 4-7: Evaluation environment for virtually integrating three DBs. 64
Figure 4-8: Evaluation results of the virtual integration environment using Presto with three methods: query pushdown, data placement analysis, and intermediate data relocation. 68
Figure 4-9: Evaluation results of query pushdown in 3DB-DT 69
Figure 4-10: Evaluation results of data placement analysis by comparing 3DB-DT and 3DB-PA. 70
Figure 4-11: Evaluation results of intermediate data relocation on 3DB-PA. 70
Figure 4-12: Execution time comparison of a single DB and data virtualization with three methods by the evaluation queries except for query 8. 74
Figure 5-1: VRP with two vehicles and four locations. 79
Figure 5-2: Binary values in a set of decision variables X in Figure 5-1 (b). 80
Figure 5-3: Execution time on D-Wave 2000Q 87
Figure 5-4: Total travel cost in small-scale problems. 89
Figure 5-5: Total travel cost in middle-scale problems 89
Figure 5-6: Execution time on Ising machines and conventional classical methods.

..... 91
Figure 6-1: An example of grouping optimization in collaborative learning. 97
Figure 6-2: Heatmaps of feasible solution rates with varied penalty coefficients. 103
Figure 6-3: Heatmaps of average objective values with varied penalty coefficients. 104
Figure 6-4: Performance comparison among QA, SA, and HC. 105

List of Tables

Table 3-1: Costs in a query Q as an example. 29
Table 3-2: Hardware environments of our prototype system 33
Table 4-1: Data placement by data type and data size in each DB. 65
Table 4-2: Information on 12 evaluation queries: type and number of tables to be used, join process across DBs by data placement, query characteristics (purpose, processing details, number of results) 66
Table 5-1: Scale of VRP problem in evaluations (V: # of vehicles, P: # of locations, S: max values of visit order). 85
Table 5-2: Penalty parameters of each annealing machine. 86
Table 6-1: Symbols to formulate the grouping optimization problem. 99

Chapter 1
Introduction

This chapter provides an overview of this thesis. In recent years, pioneering systems that utilize a variety of data via machine learning have been actively studied along with buzzwords, "big data" and "AI". These systems are called data-driven systems, and are expected to be utilized to solve various problems including social issues. This study focuses on combinatorial optimization problems as a more realistic application on the data-driven systems. First, this chapter describes the current status of the data-driven systems to solve combinatorial optimization problems as background and issues in implementing the data-driven systems for solving such problems. Then, the approach and contribution of this research are described.

1.1 Background

A combinatorial optimization problem is a search problem for an optimal solution that satisfies constraints from a large number of alternatives [1]. Examples of combinatorial optimization problems include traveling salesman problem [2], shortest path problem [3], coloring problem [4], knapsack problem [5], and scheduling problem [6]. Many of these problems are classified as NP-hard that is known to be difficult to solve in polynomial time [7]. These combinatorial optimization problems exist in various general problems. Examples of familiar problems include finding a route from a current point to a destination, creating a work shift at a workplace, and planning diversified investments of financial assets. Social problems include problems directly related to the Sustainable Development Goals (SDGs), such as reducing CO2 emissions by eliminating traffic congestion and improving the efficiency of logistics delivery plans, planning evacuation routes and disaster relief supplies in the event of a disaster. Therefore, solving combinatorial optimization problems in a realistic time is an important issue.

To solve a combinatorial optimization problem, the first step is to define the problem setup to clarify its optimization objectives and conditions. The optimization objectives are defined as a minimization or/and maximization problem, and the conditions for a correct solution are defined as constraints. For example, in the traveling salesman problem, the objective is to minimize the total travel time, and the constraint is to visit all cities only once. Next, data has to be collected to represent these objectives and constraints. In some cases, this data is simply obtained from a database system, while in other cases, the data is computed by aggregation, analysis, or a machine learning model. The data is then used to construct a mathematical model of the problem to be solved and search for the optimal solution that satisfies the constraints.

As the size of a combinatorial optimization problem increases, the amount of data and computation required to solve it grows exponentially. For example, in the traveling salesman problem, the number of cities to be traveled is a factor that determines the size of the problem, and the number of candidate travel routes and their route information increases exponentially. This is commonly called a combinatorial explosion. In addition, it is necessary to handle many types and huge amount of data in order to accurately represent the problem to be solved. In the above example, information of the travel way and real-time route information, such as congestion, is required to accurately calculate the travel time between cities. Therefore, in order to solve a realistic combinatorial optimization problem, it is required to construct a data-driven system that efficiently collects necessary data and derives solutions of its problem at high speed.

1.2 Related Paradigm

An end-to-end data-driven system, that accumulates necessary data, constructs a problem to solve with the data, and searches for a solution, has not yet been proposed for solving combinatorial optimization problems to our knowledge. However, it is possible to plan a data-driven system built by the traditional and general concept of system implementation. Figure 1-1 illustrates a diagram of a data-driven system conventionally implemented for solving combinatorial optimization problems, consisting of a data platform and a combinatorial optimization application. The data platform consists of various databases, depending on the application and data type. The combinatorial optimization application obtains data from the data platform on the

application server, creates the data necessary for optimization by data processing or machine learning as needed, and then executes a combinatorial optimization solver or an algorithm implementation to calculate the solution.

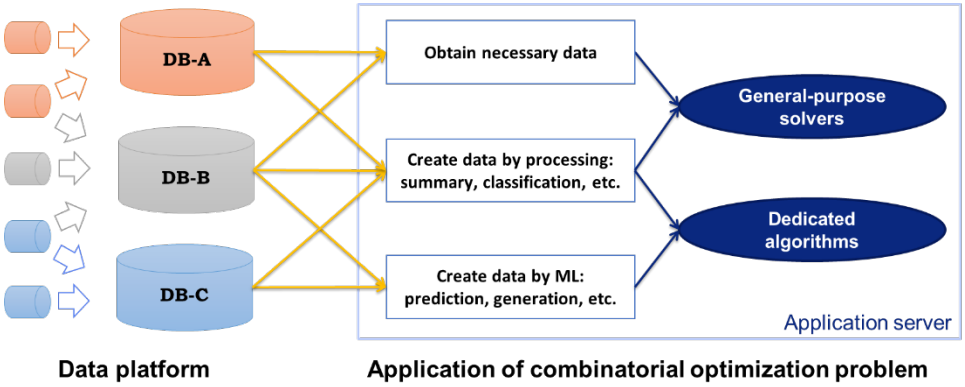


Figure 1-1: Traditional data-driven system architecture for combinatorial optimization problem.

Conventional data platforms are individually constructed according to the source of the data and the purpose of it to store the data. In recent years, a variety of physical events has been converted into data with the development of computer systems and the Internet technologies. In addition, a large scale and wide variety of data, which is called "Big Data," has been generated with the development of web services and the spread of IoT devices. As a result, it has become difficult to manage and utilize data using conventional data platform construction methods. Therefore, an integrated big data platform that integrates all data into a single system, such as a distributed database or a distributed processing system, has been proposed [8].

Specific architectures that have been proposed include data warehouses, data lakes, and lambda architectures. A data warehouse [9] is a system that integrates copies of data held by various databases into a single database in order to integrate these databases. This is specialized for data use such as data analysis, and enhances the convenience of the data use by data cleansing and summarization at the time of data collection. Data lake [10] is a system that eliminates data processing during data collection in data warehouse as much as possible and integrates not only structured data such as tables but also unstructured data such as text files into a single database. The lambda architecture [11]

is a system that consists of a speed layer that incorporates stream processing and a batch layer with a data lake for the purpose of utilizing data with low latency for larger data.

In solving combinatorial optimization problems, since it is difficult to derive the best solution (exact solution) among all candidate solutions when the number of combinations increases explosively, methods for finding approximate solutions have been developed. It is important to improve both computational speed and solution optimality for the approximate solution methods. There are two main types of the implementations. One is a general-purpose approximate solver, which can solve a combinatorial optimization problem approximately by formulating its problem into a mathematical model that the solver can handle. An example is the Mixed Integer Programming (MIP) solver [12] that can solve problems expressed in linear form. The other is a method that implements an approximate algorithm for a specific combinatorial optimization problem. Typical approximate algorithms include the greedy algorithm [13], local search algorithm [14], genetic algorithm [15], and reinforcement learning [16].

1.3 The Problem

It is sometimes difficult to build a single, completely integrated big data platform for large business, and there are multiple partially integrated databases in real business environments, such as the telecommunications business. The reasons for this include differences in the purpose of use of the data to be integrated, differences in the timing of data generation and use, differences in the type of data, and differences in the organization managed the data. As a result, the time required to gather scattered data has not yet been improved, and no effective system configuration has been proposed. This problem is particularly pronounced in combinatorial optimization problems because of the wide variety of data required to represent and formulate the problem to be solved.

Furthermore, applications of combinatorial optimization require a very large amount of computational resources depending on the scale of the problem due to the combinatorial explosion. This is no exception for use of approximate solution methods, which use a large number of fast computers to achieve high speed and high-quality solutions. Improving the performance of individual computers is required for quickly computing good solutions of a combinatorial optimization problem and for solving larger problems. However, the peak of the performance improvement is reaching due to the

limitations of narrow-line semiconductors [17]. Therefore, it is necessary to develop next-generation computers that differ from current computer architectures and methods for solving combinatorial optimization problems on them.

1.4 Research Approach

This research aims to realize a data-driven system that accelerates both the collection of data necessary for solving combinatorial optimization problems and the derivation of their solutions. The proposed data-driven system consists of a data platform that virtually integrates multiple databases through data virtualization and quantum annealing that performs the optimization process for combinatorial optimization applications. In order to accelerate the collection of data required for solving the combinatorial optimization problem, a virtually integrated data platform utilizing data virtualization is constructed as a means of integrating multiple databases. Furthermore, quantum annealing is utilized to solve large-scale combinatorial optimization at high-speed using fewer computational resources to accelerate the derivation of solutions to combinatorial optimization problems.

To realize the proposed data-driven system for combinatorial optimization problems, we aim to advance both data virtualization and quantum annealing. In terms of data virtualization, there is an issue of inferior performance of its query processing compared to that of physical integration, and we study methods to minimize this issue. Specifically, we study data virtualization for query engines sharing data and data virtualization for dividing a large database. In terms of quantum annealing, there is an issue that its performance in actual combinatorial optimization problems is uncertain. We consider evaluating the practicality of quantum annealing by comparing it with classical methods in actual problems. Specifically, we study the application of quantum annealing to the vehicle routing problem and the grouping optimization problem.

1.5 Contributions

- As a data-driven system for solving combinatorial optimization problems, we proposed a system architecture that achieves both faster data collection through data virtualization and faster solution derivation through quantum annealing.

This enables data integration, which has been difficult in real business environments, and reduces the time required to search and collect the data needed to solve a combinatorial optimization problem from a database system. In addition, the time required to solve the problem is also reduced, making it possible to apply the system to realistic problems.

- We proposed a data virtualization system to achieve efficient query processing for multiple query engines that share data in same storage. This addresses the issue of query processing in data virtualization, which is inferior to physical data integration for multiple query engines that share data. Our evaluation by its prototype system shows that it accelerates queries by automatically selecting the appropriate query engine per query operator from multiple query engines on the same storage.
- We proposed a data virtualization system that achieves virtual integration for dividing an existed database. The system is equipped with three methods that can solve communication bottlenecks in data virtualization: data placement analysis, query pushdown, and intermediate data relocation. Through quantitative evaluation with its prototype system using real data, we show that these methods can be used to divide and virtually integrate a large database system with minimal communication bottlenecks in data virtualization.
- We quantitatively evaluated the effect of quantum annealing on the vehicle routing problem, which has many applications as a combinatorial optimization problem, by comparing real quantum annealing machine with classical methods using data that assumes real service use. Based on the evaluation results, we discussed the applicability to large-scale problems and summarized the issues and conditions for utilizing the quantum annealing machine.
- We formulated a general model for the grouping optimization problem, which assumes cooperative learning as a combinatorial optimization problem on this data-driven system. And we quantitatively evaluated the effect of quantum annealing by using real quantum annealing machine.

1.6 Organization of Thesis

First, Chapter 2 describes the system architecture of the proposed data-driven system.

This also describes the technical elements related to data virtualization and quantum annealing, which are the components of the system, and the issues to be addressed in this research. Furthermore, this describes the use cases of the proposed data-driven system.

Chapter 3 describes a data virtualization system for query engine sharing same data, and explain the virtual schema and query splitting method that constitute the system. We evaluated the effectiveness of the proposed system by implementing a prototype on Hadoop HDFS as shared storage, SQL-on-Hadoop Hive and Impala as multiple SQL query engines, and Solr as a full-text search engine.

Chapter 4 describes the functions that data virtualization should provide for the purpose of operating a large database system by dividing it. To solve the communication bottlenecks in data virtualization when handling large data, this study quantitatively compares and evaluates query processing performance of a data virtualization system implemented with three methods: a query pushdown method, a data placement analysis method, and an intermediate data relocation method.

Chapter 5 evaluates quantum annealing on a basic VRP in order to reveal the effect of quantum annealing on a VRP with the scale of a demand transportation service. Based on the evaluation results, this chapter also discuss issues and conditions for utilizing quantum annealing in VRPs assuming real-world applications.

Chapter 6 formulates a general model of the grouping optimization for quantum annealing, and evaluate it assuming collaborative learning on quantum annealing machine in comparison with simulated annealing and hill climbing. As a result, we confirmed that quantum annealing solves this problem about 400 times faster than the hill climbing and about 1,300 times faster than simulated annealing, although the stability of solution’s accuracy on quantum annealing machine is lower than these methods.

Finally, chapter 7 concludes our research on data-driven systems for solving combinatorial optimization problems. In particular, this also describe the remaining issues related to data virtualization and quantum annealing, which are the technologies that comprise the data-driven system, as well as the system implementation of machine learning, which is an important component of the proposed data-driven system.

Chapter 2
A Data-Driven System for Combinatorial Optimization Problems

This chapter describes the system architecture of the proposed data-driven system for combinatorial optimization problems. This data-driven system composes a data platform with data virtualization and an application using quantum annealing as a solver of combinatorial optimization problems in order to improve performance of both data collection and solving the problems. This chapter provides an overview of both data virtualization and quantum annealing, and the issues to be addressed for these technologies in this research.

2.1 System Architecture

The proposed data-driven system accelerates both the collection of data needed to solve a combinatorial optimization problem and the derivation of its solution. The architecture of the system is shown in Figure 2-1. The system consists of a data platform and a combinatorial optimization application.

The data platform is virtually integrated of multiple databases by data virtualization. Here, we assume that the data necessary to construct a model of a combinatorial optimization problem is stored in multiple databases. The user, including the system solving the combinatorial optimization problem, accesses these data via the data virtualization system. Thus, by virtually integrating data scattered across multiple database systems, it reduces the time required to search for data and eliminates the effort required on the part of the user, such as differences in interfaces between database

systems. In addition, since there is no need to physically obtain and manage a copy of the data from the data source, not only it makes system construction easier, but also data consistency management is no longer required.

The combinatorial optimization application consists of an application server that builds models of combinatorial optimization problems that can be executed by quantum annealing, and quantum annealing that actually solves the combinatorial optimization problems. This enables fast solving of large combinatorial optimization problems at high speeds and requires fewer computational resources on the application server. The application server obtains the necessary data through a data virtualization system to build its model. In some cases, the acquired data is processed through analysis to create the data necessary for the combinatorial optimization problem. In other cases, the acquired data can be used to predict something to create the necessary data by using machine learning models trained in advance.

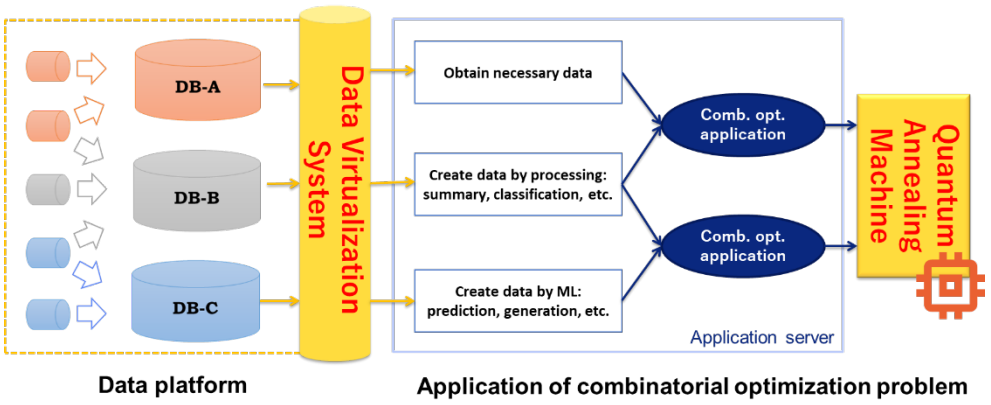


Figure 2-1: Proposed data-driven system architecture for combinatorial optimization problem, which includes data virtualization system in data platform and quantum annealing machine for solving combinatorial optimization.

2.2 Data Virtualization

Data virtualization is a technology that virtually integrates multiple data sources (DS) as a single database system [18]. In virtual integration by data virtualization, only the schemas of the target DSs are integrated to provide a single integrated database system

to users while keeping the data in the existing DSs. This enables users to quickly access data without having to search for the DS to obtain the desired data. In addition, joining data among multiple DSs can be handled with a single query. The target DS can be any data system that has an interface for providing data, not just a database system with an SQL interface. For example, file systems that support file access via networks such as FTP, and web servers that can access web data via HTTPS, etc. are also included.

2.2.1 Components of Data Virtualization System

The data virtualization system is implemented to provide a mechanism to virtually integrate multiple DSs as data virtualization. Users of the data virtualization system can access data in the virtual integrated DSs via SQL. This SQL query, as well as a query to a regular database management system (DBMS), specifies and processes the data to be accessed as a table. Therefore, the data virtualization system implements a SQL query processing engine that includes a query parser and query optimization, just like a regular DBMS. The major difference from a regular DBMS is that the data is on a different system and the query processing for the data is based on this situation.

The management of data on different systems requires the management of information about the DS where the data resides, in addition to the usual data catalog in a DBMS. The data source information includes data formats and connection protocols for obtaining the target data from that DS.

The query processing of the data virtualization system uses this DS information to connect to the target DS and obtain data. The connection to the DS is implemented as a connector using the connection protocol provided by the DS. When obtaining data, a query to obtain data is generated according to the interface of the DS, and this connector is used to connect to the DS and execute the query. For example, if the target DS supports JDBC, the JDBC connector is used to generate SQL statements when executing the query process to acquire data. In addition, depending on the target DS, the data format may not be the table format handled by the data virtualization system, and the data must be converted.

2.2.2 Issues of Data Virtualization

Data virtualization is a virtual integration of multiple DSs that allows users to access data without being aware of the location of the data and the interface of the DSs to which

they are connected. However, data have to be obtained from DS each time a query is executed. This communication between data virtualization system and DS reduces query processing performance compared to physical integration.

In this study, we improved query processing performance in data virtualization system by reducing communication, which is the bottleneck of data virtualization, in two situations. One is to assume the case of target consolidation in an environment where multiple target DSs are managing data in shared storage. This study is described in Chapter 3. The other is a case where one huge DS is divided into multiple DSs and virtualized through data virtualization. This study is described in Chapter 4.

2.3 Quantum Annealing

Quantum annealing [19] can search for solutions to combinatorial optimization problems at high-speed using the principles of quantum mechanics, and has attracted attention as a fast metaheuristic method. Recently, hardware implementations of quantum annealing [20] have become commercially available, and its use as a kind of quantum computer is being considered in various combinatorial optimization problems.

2.3.1 How to Use

Quantum annealing is a technique that uses the principles of quantum mechanics to search for the ground state of an Ising model. This ground state can be assumed to be the optimal solution to a combinatorial optimization problem, and quantum annealing can search for the optimal solution by mapping the combinatorial optimization problem to this Ising model. The Ising model is expressed in terms of the interaction coefficients J_{ij} for two spins $\sigma_i, \sigma_j \in \{-1, +1\}$ and an external field energy coefficient h_i affecting a spin σ_i as follows:

$$\mathcal{H} = \sum_{i,j} J_{ij}\sigma_i\sigma_j + \sum_i h_i\sigma_i \,. \tag{2-1}$$

Quantum annealing searches for the spins that minimize the Hamiltonian $\mathcal{H}$ as this ground state. Therefore, by formulating the combinatorial optimization problem as a minimization problem in this Ising model, quantum annealing allows a fast search for a good solution. To make the formulation of the combinatorial optimization problem intuitive, the spins σ_i can be transformed into binary variables $x_i \in \{0,1\}$ as follows:

$$x_i = \frac{\sigma_i + 1}{2}. \tag{2-2}$$

The following Quadratic Unconstrained Binary Optimization (QUBO) Problem [21] is acquired by applying this to the Ising model in Eq. (2-1).

$$\mathcal{H} = \sum_{i,j} Q_{ij} x_i x_j \,. \tag{2-3}$$

In general, a combinatorial optimization problem has not only an objective function to be optimized, but also soft constraints that should be followed as much as possible and hard constraints that must be followed as constraint conditions. The penalty method in which constraint violations are added to objectives as penalties is used to formulate them in QUBO. Parameters are set for each constraint, and the parameters are tuned so that the constraint is not violated as much as possible. In particular, hard constraints are set with large parameters, while soft constraints with low priority are set with small parameters. Here, a solution which satisfies the hard constraints is called a feasible solution. If the parameter is too large, changes in the value of the objective function will not be affected, and good solutions will not be obtained among the set of feasible solutions. Therefore, it is important to tune the parameters so that good solutions can be obtained while feasible solutions are obtained with high probability.

Quantum annealing is a heuristic solver for combinatorial optimization problem, and the solution obtained from it is an approximate solution, not always an optimal solution. In addition, since the approximate solution does not always satisfy the constraints, it has to be run multiple times to output several solutions. After checking for violations of the required constraints, the solution with no constraint violation and better objective values is selected for use.

2.3.2 Issues of Quantum Annealing

Quantum annealing, as a kind of metaheuristic solver, is expected to efficiently solve combinatorial optimization problems that are difficult to solve with classical computers. Toward the practical application of quantum annealing, some studies have been emerged to evaluate its effectiveness by comparing it with various classical methods, such as the shift scheduling problem [22], the portfolio optimization problem for financial assets [23], and the shared spectrum allocation problem for 5G networks [24]. However, there are still few studies like this that construct the QUBO formula in real problems and

compare it quantitatively with classical methods, thus its practicality in various combinatorial optimization problems has not yet been evaluated.

This study investigates the vehicle routing problem and the grouping problem as combinatorial optimization problems that have a wide range of applications and for which the effectiveness of quantum annealing has not yet been evaluated. Specifically, we formulate these problems in QUBO and compares quantum annealing and classical methods on real-world data. Chapter 5 discusses the delivery planning problem and Chapter 6 discusses the grouping problem.

2.3.3 Related Technologies

2.3.3.1 Ising Machines

Similar to quantum annealing, Ising machines that utilize quantum inspired techniques are also attracting attention as machines for solving combinatorial optimization problems. Ising machines are implementations of classical methods of quantum annealing (e.g., simulated annealing) with software running on a classical computer or classical hardware. Although it is inferior to quantum annealing in terms of speed, it is expected to provide a more stable and better approximate solution. In addition, since the current implementation of quantum annealing has significant limitations on the scale of applicable combinatorial optimization problems, Ising machines may be utilized from a practical standpoint in applying it to real-world large-scale combinatorial optimization problems. Ising machines, as with quantum annealing, can be utilized by formulating combinatorial optimization problems into QUBOs, which allows for a relatively easy migration to quantum annealing. Here, the discussion will focus on the application of quantum annealing in anticipation of a larger scale of applicability with the evolution of its hardware implementation, but Ising machines will also be used as a comparison target for evaluations.

2.3.3.2 Gate-type Quantum Computers

The quantum annealing machines currently implemented in the market are so-called analog quantum computers and cannot be used for applications other than combinatorial optimization problems. On the other hand, quantum computers for use in general-purpose applications are called gate-type quantum computers, and their research and development are underway. In a gate-type quantum computer, operations for qubits are

performed by quantum gates. The algorithm for computing using a gate-type quantum computer is called a quantum algorithm. A quantum algorithm is implemented by a quantum circuit that consists of an array of quantum gates. Quantum algorithms include the Grover algorithm [25] for database searching, the Shor algorithm [26] for finding integer prime factors, the QFT (Quantum Fourier Transform) [27] for computing discrete Fourier transforms, the HHL algorithm [28] for solving simultaneous linear equations, and many others have been proposed. Many of these proposed quantum algorithms have been theoretically proven to be exponentially faster than classical computers, called quantum speedup.

Many available implementations of gate-type quantum computers have already been released. Examples include IBM Quantum [29] and Google Sycamore [30]. However, these are known that it is difficult to apply the above quantum algorithms to realistic problems with current implementations of these quantum computers. The reasons for this are the small number of qubits and the fact that the quantum algorithms do not work as expected due to errors in qubits and quantum gates, called quantum error. Current quantum computers that are small and affected by quantum errors are called NISQ (Noisy Intermediate Scale Quantum) devices [31]. To solve these problems on NISQ devices and to run quantum algorithms for realistic problems, research and development of Fault-Tolerant Quantum Computers (FTQC), in which qubits are sufficiently increased and a quantum error correction method is implemented, is underway. On the other hand, NISQ algorithms that allow quantum errors and operate with fewer qubits are also being researched and developed in parallel as algorithms to utilize these NISQ devices. The mainstream NISQ algorithm is a hybrid algorithm that combines a quantum computer with a classical computer, such as QCL (Quantum Circuit Learning) [32], which realizes neural network-based supervised machine learning, and VQE (Variational Quantum Eigensystem) [33], which searches for the ground state of a material. In contrast to the NISQ algorithm, the original quantum algorithm is also called the Long-term algorithm.

Quantum algorithms for solving combinatorial optimization problems also exist. The Quantum Approximate Optimization Algorithm (QAOA) [34] is a NISQ algorithm that can solve combinatorial optimization problems using the energy of the Ising model as a cost function, similar to quantum annealing. It is a hybrid classical quantum algorithm in which the values of the parameters that minimize the cost function are computed by a

classical computer. The VQE mentioned above can also be used to solve combinatorial optimization problems by searching the ground state as a minimization of the cost function in the same way. As long-term algorithms for solving combinatorial optimization problems, dedicated quantum optimization algorithms[35][36] or the use of the quantum search algorithms, Grover algorithm [37] and quantum walk [38][39], have also been considered.

The quantum algorithms for solving these combinatorial optimization problems are more difficult than Ising machines and quantum annealing to solve realistic combinatorial optimization problems at this stage due to the hardware implementation problems of quantum computers, especially the small number of qubits. On the other hand, QAOA and VQE, which are NISQ algorithms, and Grover algorithm, which is a long-term algorithm, can be used by formulating the combinatorial optimization problem into QUBO, which is the same as Ising machines and quantum annealing. Therefore, some of the knowledge gained from the utilization of quantum annealing can be diverted to the utilization of quantum algorithms. Based on the above, this paper focuses on the utilization of quantum annealing.

Chapter 3
Data Virtualization for Query Engines Sharing Data

Recently, new architecture of the data source has been developed, which can use multiple query engines on a shared storage. These query engines (QEs) share same data, but have different characteristics each other. It includes the Hadoop and its eco-systems as a typical example. However, a multidatabase system cannot automatically select a QE to post a query to these QEs sharing same data. As a result, the users must use different QEs in spite of a logical single database system. This leads additional burdens of users. And in worst case, it occurs a problem that it leads system crash by the inappropriate selection of a QE. This chapter describes a data virtualization system for query engine sharing same data, and explain the virtual schema and query splitting method that constitute the system. We evaluated the effectiveness of the proposed system by implementing a prototype on Hadoop HDFS as shared storage, SQL-on-Hadoop Hive and Impala as multiple SQL query engines, and Solr as a full-text search engine.

This chapter is organized as follows: Section 3.1 provides an overview and use cases of the proposed system, Section 3.2 describes the issues of previous studies, Section 3.3 provides details of the proposed system, Section 3.4 describes the prototype implementation of the proposed system, and Section 3.5 evaluates this prototype system. Finally, Section 3.6 presents the conclusions of the proposed system and future work.

3.1 Introduction

3.1.1 Background

Various types of data are being generated, such as Web data and log data generated by electronic devices. Ideally, these data should be consolidated into a single data source

(DS) for utilization, but in reality, the number of DSs increases for different purposes. Therefore, the importance of rapid data integration based on the premise that there are multiple DSs is increasing. A data virtualization system [18] (or multi-database system[40]) can address this issue, and can treat multiple DSs as a single DS logically. This enables rapid data utilization without modifying existing DSs.

In general, a single DS consists of a set of storage and a QE. For example, a database system consists of storage and DBMS. On the other hand, there is a new architecture in which storage and QE are separated, and multiple QEs with different characteristics can be used for data on a shared storage. Each QE shares the same data and is optimized according to the type of data and processing. However, conventional data virtualization systems do not support an environment where multiple QEs have the same data as described above in the target DS. It recognizes that there are as many DSs (sets of storage and QEs) with the same data. Therefore, DSs cannot be virtually hidden and need to be used differently, which causes a burden on users. In addition, users may select inappropriate QEs, resulting in problems such as significant delays and system crashes.

3.1.2 Contributions

We propose a data virtualization system that automatically selects an appropriate QE from multiple QEs with the same data. The proposed system provides a virtual schema that allows multiple QEs with the same data to be recognized as a single DS. This virtual schema enables automatic selection of appropriate QEs. Furthermore, the proposed system provides a query splitting method that automatically generating queries to collaborate QEs each other according to their characteristics and executes these queries in appropriate QEs on an operator-by-operator basis. This reduces user load and provides an efficient way to use QEs. If the same data of the collaborated QEs is located in the same storage space, the storage space can be used for efficient coordination among QEs.

The contributions in this study are follows:

- By providing a single virtual schema for multiple QEs with the same data, we solved the problem of creating multiple virtual schemas.
- The query splitting method using the above virtual schema enables automatic selection of an appropriate QE from multiple QEs with the same data, thereby reducing the burden on the user.
- The query splitting method enables the generation of split queries by selecting

appropriate QEs for each operator of the original query, and achieves query speedup by utilizing multiple QEs.

3.1.3 Use Cases

A typical system architecture for the proposed system is Apache Hadoop[41] as shown in Figure 3-1. Hadoop can store data in file format and can integrate data of various sizes and in various formats into a single system. In order to realize query processing for such data, many QEs with different characteristics have been developed on Hadoop Distributed File System (HDFS) as a common storage. An example of a QE characterized by the scale of data size is SQL-on-Hadoop, which can execute MapReduce using HiveQL, a SQL-like language. Apache Hive [42] is one of the SQL-on-Hadoop implementations, which is appropriate for processing large data by using MapReduce. On the other hand, Impala [43], Apache Drill [44], and Presto [45], which do not use MapReduce but specialize in SQL processing and perform on-memory processing, are capable of high-speed processing of small data that can fit in memory. There are also various types of QE that specialize in different types of data and processing, such as column-oriented data stores (Apache HBase [46], Apache Kudu [47]), DAG query processing with dependency relations (Apache Pig [48]), full-text search engine (Apache Solr [49]), query processing for graph data (Apache Giraph [50]), and so on. By using the proposed system in an environment with these multiple QEs, the user can select the appropriate QE without being aware of the characteristics of each QE.

There are two possible use cases of analysis using the proposed system: analysis in an environment where multiple data with different data sizes are mixed, and analysis in which processes with different characteristics are mixed. In the former case, for example, as in purchase forecasting, QE for large-scale data (e.g., Hive) is used to predict the preferences of each customer by linking the purchase history, which is large-scale data, with customer information, while QE for small-scale data (e.g., Impala) is used to link products related to the preferences of each customer from product masters. This analysis can be performed with a single query across these QEs on the proposed system. In the latter type of analysis, for example using Twitter data, a full-text search QE can extract users who say specific phrases (e.g., "radio" and "bad"), and a QE with an aggregation function (e.g., Impala) can aggregate the number of results and followers for each user to calculate the degree of influence of that phrase on Twitter. The proposed system can

thus perform analysis with a single query as well as the former case. In this way, the proposed system can automatically select the most appropriate QE for each data or execute the query when multiple data or processes with different characteristics are mixed in a single query. This is expected to reduce the burden on the user and to be faster than execution with a single QE.

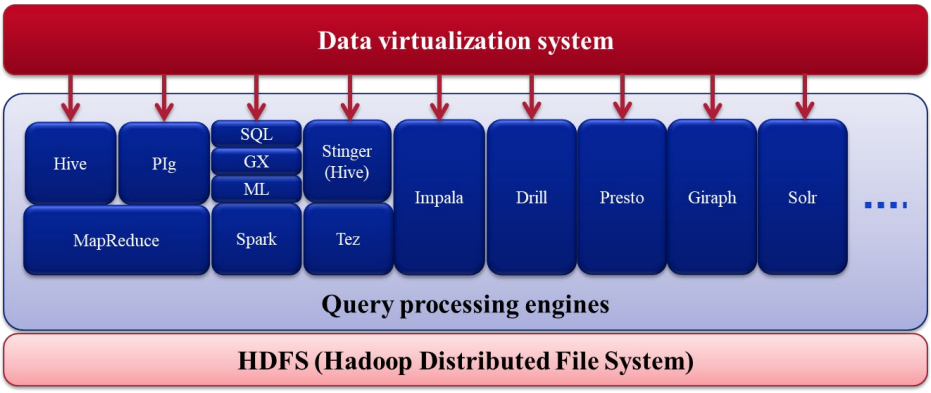


Figure 3-1: A use case of our proposal system.

3.2 The Problem

3.2.1 Issues of the Data Virtualization System

Research on multi-database systems that integrate multiple DBMS has long been conducted, and many prototype systems (HERMES [51], Pegasus [52], TSIMMIS [53], Garlic [54], etc.) have been developed for RDBMS [55][56] . In recent years, data virtualization systems (e.g., GReIC [57], OASIS [58], ViDa [59], Teiid [60], etc.) that provide integrated schemas not only for RDBMS but also for various DS have been proposed. These conventional data virtualization systems configure virtual schemas on a one-to-one basis for the DS to which they are connected using the GAV method [61]. However, in an environment where there are multiple QEs with the same data, as shown in Figure 3-2 , a virtual schema with the same schema is created for each connection point for the same data. To execute a query in this environment, the user must manually select the appropriate virtual schema for the QE from the multiple virtual schemas. As a result, the user is required to understand the characteristics of each QE and to match the

characteristics of the appropriate QE with the query to be executed. In addition, the query-by-query QE selection method does not maximize the performance of an environment with multiple QEs with different characteristics, because it assigns a query to one of the QEs even if there are multiple processes with different characteristics in a single query. Based on the above, the following issues should be solved by a data virtualization system in an environment with multiple QEs that have the same data.

- Multiple virtual schemas for the same data
- Manual selection of appropriate QE
- Query-by-query QE selection

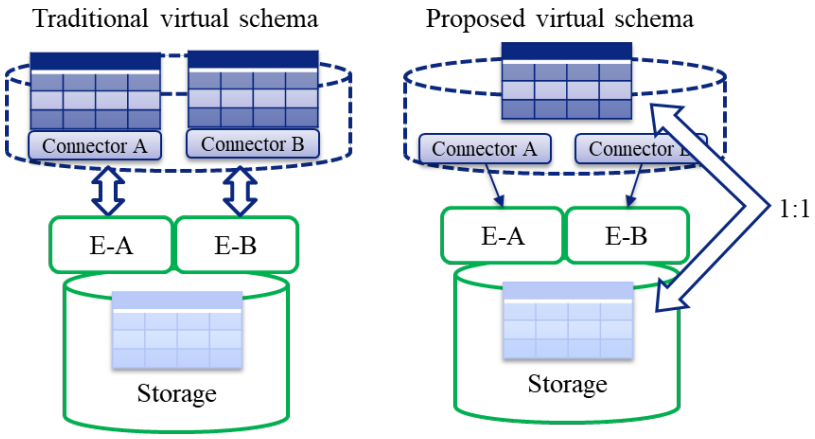


Figure 3-2: Virtual schema: existing system and proposal system.

3.2.2 Related Work

HadoopDB/Hadapt[62] places an RDBMS (PostgreSQL) on each node as a Hadoop data node, performs distributed processing by converting SQL queries to MapReduce, and reconverts the processing on each node to SQL for execution. This provides the functionality of a data virtualization system via Hadoop, treating the same data as replication and solving the virtual schema issue. However, it does not assume an environment where there are multiple QEs with different characteristics, so the appropriate QE must be manually selected in advance.

Polybase [63] is a query splitting system for Hadoop and RDBMS, which converts processes to HDFS data in SQL queries into MapReduce processes. This enables Hadoop

to be used as a single database system. However, if Hadoop has multiple QEs that handle the same data, the same issues as conventional data virtualization systems arise.

DBMS+ [64] and HFMS [65] propose the concept of a system that separates multiple target systems by QE and storage, and selects more suitable QE and storage depending on the query. DBMS+ is research on the execution of CQL (Continuous Query Language) queries on multiple streaming systems on Hadoop as QEs at appropriate QEs in response to user requests. HFMS is a study on workflow optimization of multiple queries across multiple QEs. Both of them enable automatic selection of appropriate QEs, but they do not solve the problem of selecting QEs on a query-by-query basis.

3.3 The Proposed System

3.3.1 Overview

The proposed system provides the following features to solve the three problems of conventional data virtualization systems.

- A single virtual schema for the same data.
- Automatic selection of appropriate QE.
- Operator-by-operator QE selection by splitting queries.

This will speed up query execution while reducing the load on the user. Figure 3-3 shows the system architecture for executing split queries on multiple QEs with the same data using the proposed system. Clients submit queries to the proposed system in the same way as a stand-alone database system with a single virtual schema. The proposed system executes the query by selecting an appropriate QE for the submitted query based on the query splitting method. If it judges that each operator in the query is suitable for a different QE, it generates split queries and executes these queries with the appropriate QEs. To achieve this, the query splitting method consists of query splitter that decomposes the query into operator units and assigns an appropriate QE to each of them, and query execution controller, which executes the generated split queries in an appropriate order. Thus, the proposed system allows the users to transparently and automatically select appropriate QEs, generates split queries, and assigns each query to an appropriate QE for efficient query processing.

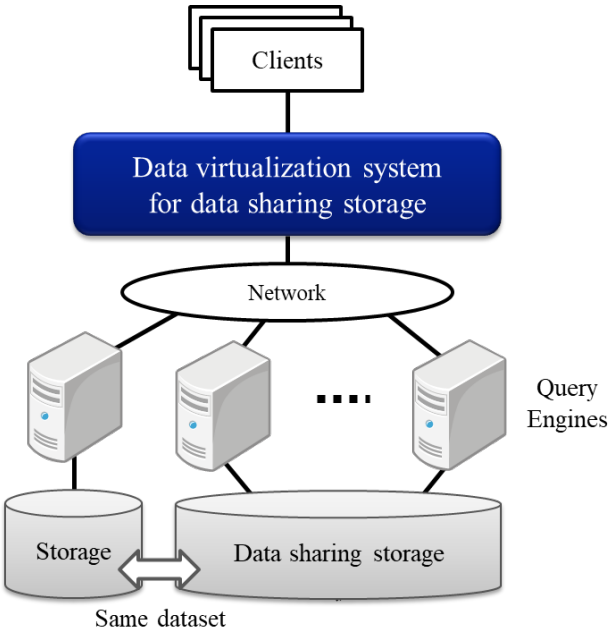


Figure 3-3: System architecture of proposed data virtualization system for data sharing storage.

In this architecture, when multiple QEs share the same data on storage, there is no need to transfer intermediate data over the network whenever a QE is switched. This enables low latency switching of the QEs. To realize such QE switching utilizing shared storage, it requires a virtual schema recognizing that the target QEs share the same storage, and split query executor including query splitter and execution controller that utilizes shared storage when switching to multiple QEs. Figure 3-4 shows an example of this. In this example, three QEs execute as follows:

(a) Write the result of processing X to storage as Temp1.

(b) Write the result of processing Temp1 to storage as Temp2.

(c) Return the result of processing Temp2 and Y to the data virtualization system as the final result.

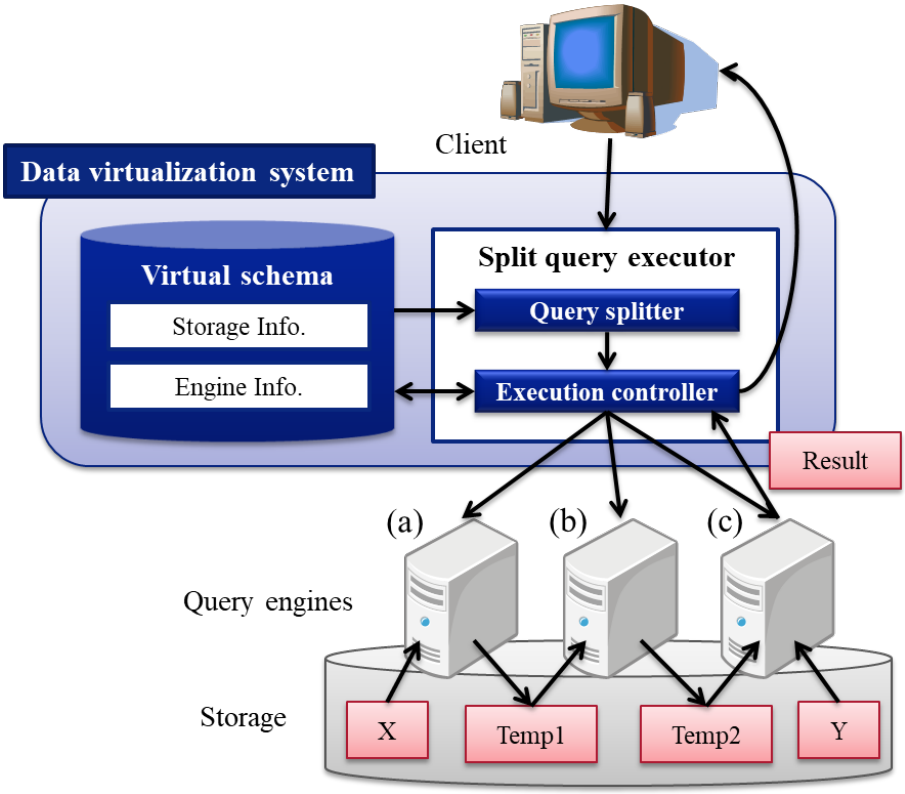


Figure 3-4: System components and an example of executing a split query.

The following sections describe the target queries and QEs as well as the assumptions of the proposed system, and describe the virtual schema and the query splitting method to realize the proposed system.

3.3.2 Assumptions

The proposed system splits a query into units of operators in the query plan. The operators themselves are not split, and similarly, the data to be processed is not split. In addition, the execution order of the split query is controlled to ensure consistency of query processing results before and after the split is applied.

The query language, which is the user interface of the proposed system, only needs to be made clear to the operator by query plans for query splitting. Therefore, the query splitter is available even with a proprietary query language if its query parser can generate query plans. This means the query splitter is independent of the specific query language in this condition. For example, the operators in a SQL query includes projection

operator, join operator, and modules that cannot be further decomposed, such as aggregation functions and UDFs.

The target QEs are ones that can execute operators defined in the implementation of the proposed system with predefined queries such as SQL or commands. Therefore, the translation rules between the operators defined in the proposed system and the queries of the target QEs need to be defined in advance. In addition, the target QEs must share the same data, and it is assumed that data updated by one QE will be immediately reflected in the other QEs.

There are two types of QE combination patterns: one is a combination of QEs whose availability and performance differ depending on the amount of used resources, and the other is a combination of QEs whose availability and performance differ for specific data or operators. The former is applicable to a combination of QEs whose performance is lower than that of other QEs depending on the utilization of certain resources. Examples of resource utilization include the memory size used for a given query, the number of used CPU cores, and network load. The latter can be applied to a combination of QEs that are specialized for a particular operator and are faster than other QEs. The combination of these factors can also be applied to QEs in which the performance of certain operators deteriorates under a certain resource environment.

3.3.3 Virtual Schema

In order to make users unaware of the QE to which they are connected, the proposed system links physical schemas and virtual schemas by one-to-one, which are the same data on storage, with independent on the number of QEs. For this purpose, the information of the virtual schema is separated into schema and connection information as shown in Figure 3-2. These are managed as storage information and engine information in the virtual schema shown in Figure 3-4, respectively. The storage information is provided as a schema to be shown to users. The statistical information (data size, maximum/minimum values, etc.) of the target data is also stored as the storage information. The engine information is stored as the physical connection information between the proposed system and the QE. Furthermore, the performance and specifications (non-executable queries, processing algorithms, etc.) of each QE process are stored and used to determine the QE that executes the process. This virtual schema prevents multiple virtual schemas from being generated for multiple QEs with

the same data, and eliminating the need for the user to select a virtual schema. This virtual schema also enables automatic selection of QEs by the query partitioning method described below.

3.3.4 Query Splitting Method

3.3.4.1 Query Splitter

This section describes the generation of a split query in the query splitter to realize the query processing of the proposed system shown in Figure 3-4. Figure 3-5 depicts the processing flow of the query splitter.

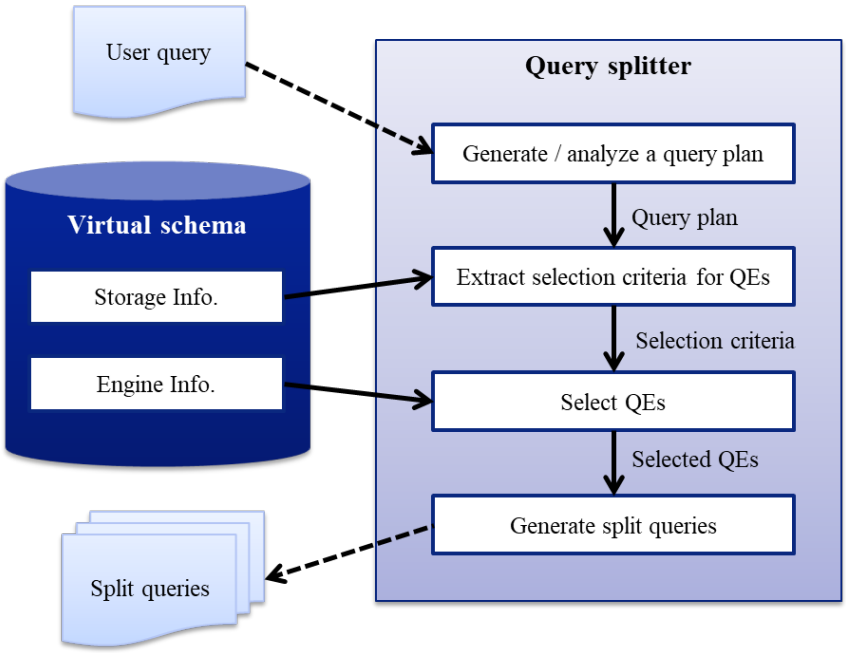


Figure 3-5: Processing flow of the query splitter.

First, the query splitter generates a query plan, which indicates the query execution plan based on the query submitted by the client, and extract the information on the query needed to determine the QE to be executed from the query plan and storage information. Since the query plan here should be generated so that the processing cost becomes smaller, we apply an existing workflow optimization method implemented in Hadoop and conventional data virtualization systems (e.g., Garlic [54], Starfish [66], Hyracks

[67], etc.). For the query plan generated here, information on resource usage, which is one of the selection criteria for determining the QE to be executed, is added to the query plan. This information is calculated for each operator of the query plan using statistical information in the storage information. The information on resource usage and engine information added to the query plan are used as the selection criteria to determine the QE to be executed for each operator. The selection criteria for QE are described in Section 3.3.4.4. Finally, split queries are generated according to the execution form and grammar of each QE. In addition, the system generates management queries necessary for switching QEs.

Split queries using multiple QEs incur split loss due to the overhead of interpreting and optimizing the query at each QE, reading and writing intermediate data when switching QEs, and so on. If the performance of the original query is degraded by generating split queries due to this split loss, it is necessary to avoid this. The query splitter avoids performance degradation due to split loss by introducing a split query performance model as described in Section 3.3.4.3.

3.3.4.2 Execution Controller

When executing split queries, each query requires intermediate data, so the next query cannot be executed until the previous query that generates the intermediate data is finished. Execution controller takes this dependency into account when controlling the execution of split queries.

First, the split queries are executed based on the execution order recorded when generating these split queries. Once a split query is executed, the execution controller waits until the query completion is notified, which indicates that the generation of intermediate data is complete. As soon as this is received, a next split query is executed. This process continues until all split queries have been executed, and the execution of all split queries are complete when the final query result is obtained. This result is analyzed to obtain the number of results and its data size, and also analyzed with the operators of the queries at the time of submission to collect storage information such as selection rates for each operator as statistical information. This information is used to calculate the amount of resources used by each operator when generating split queries and selecting QEs as selection criteria.

Here, let us discuss the failure case of query execution in QE. There are four possible

reasons for query execution failure.

(1) Estimation error of resource usage for each operator.

(2) Incorrect decision criteria for QE.

(3) System failure.

(4) Dynamic changes in the available resources by other users.

The actions to be taken in a failure of query execution differ for each situation. For the estimation error in (1) and the incorrect decision criteria in (2), it is possible to try to generate the split query again by updating the information in the virtual schema (estimation parameters, upper limit of available resources, etc.). If the updates in (1) and (2) are unknown, it is necessary to execute the original user query on a single QE that can execute it surely without splitting. If there is no executable QE, the query execution will terminate with an error. In the case of (3), the query cannot be executed, resulting in an error termination. The loss until the query fails can be avoided by conducting a communication test in advance. And if there is another QE that can execute the query, it is possible to execute the query there. On the other hand, in a multi-user environment where (4) occurs, it becomes difficult to separate it from (1) and (2), and an error termination occurs when query execution fails, including errors of (1) and (2). This is because it is not possible to distinguish whether the cause of the failure is a problem in the split query generation or a change in the amount of available resources. Note that (4) does not occur when QE uses Linux cgroups, etc., and the upper limit of available resources does not change for each user.

3.3.4.3 Performance Model

To take the split loss due to query splitting into account, we define a performance model of the split query. It is assumed that a set of QEs E holds identical data and each QE E_x, where the variable x takes $1 \leq x \leq |E|$, is connected to the proposed system. When a client submits a query Q, the proposed system divides it into a set of queries q and maps an appropriate QE E_{x_i} to each query q_i where the variable I takes $1 \leq i \leq |q|$. The final split query $splitQ$ contains the set of split queries q and a set of management queries $mqList$ such as creating a location in a shared storage for intermediate data. The performance ratio $Perf(\cdot)$ between executing the original query Q in QE E_x and the $splitQ$ generated by the proposed system can be formulated as follows:

$$Perf(Q, E_x, splitQ) = \frac{Cost(E_x,\ Q)}{C(mqList) + \sum_{i=1}^{|q|} Cost(E_{x_i},\ q_i)}, \quad (3\text{-}1)$$

where the function $C(mqList)$ represents the execution cost of management queries $mqList$, and the function $Cost(E_x,\ Q)$ represents the cost of executing query Q with QE E_x. In this section, the cost means an execution time. The denominator of the right side of Eq. (3-1) represents the total execution cost of the split queries generated by the proposed system, and the numerator represents the cost of executing a original query in a single QE without splitting.

The proposed system determines the performance of split queries using the Eq. (3-1). If $Perf(Q, E_x, splitQ) > 1$, then the performance of these split queries generated by the proposed system is higher than the performance of the original query Q executed on a single QE E_x. On the other hand, if $0 < Perf(Q, E_x, spiltQ) < 1$, then no performance improvement is expected from the query splitting.

The function $Cost(E,\ Q)$ depends on the QE implementation. It has two major components: $Cost(E,\ Q) = Init(E,\ Q) + Exec(E,\ Q)$, where the function $Init(E,\ Q)$ represents the cost of initialization such as query optimization, and the function Exec$(E,\ Q)$ represents the cost of data processing determined by the query processing algorithm and data access method. Using these functions, the performance improvement condition $Perf(Q, E_x, splitQ) > 1$ can be expanded as follows:

$$Exec(E_x,\ Q) - \sum_{i=1}^{|q|} Exec(E_{x_i},\ q_i) > \mathrm{C}(mqList) - Init(E_x, Q) + \sum_{i=1}^{|q|} Init(E_{x_i}, q_i). \quad (3\text{-}2)$$

Here, the left side is the performance improvement of query processing due to query splitting, and the right side is the overhead caused by the query splitting. By measuring or estimating each cost in advance, the performance degradation caused by query splitting can be avoided in the proposed system by evaluating Eq. (3-2).

An example of the performance model is described here. This example calculation is shown when a query Q is allocated to two QEs A and B with identical data. First, as an example of a pattern that improves performance, the query Q is split into three queries, q_1, q_2, and q_3, and a managed query $3split$ is generated. These split queries are executed by B for q_1, A for q_2, and A for q_3. The initialization process for each QE is the same regardless of the query. Assuming that the costs in Eq. (3-2) are shown in Table 3-1, then the left and right sides of Eq. (3-2) are respectively as follows:

$$Exec(A,\ Q) - \sum_{i=1}^{3} Exec\left(E_{x_i},\ q_i\right) = 297.34,$$

$$C(3split) - Init(A,\ Q) + \sum_{i=1}^{3} Init\left(E_{x_i},\ q_i\right) = 4.55.$$

These calculations indicate that the left side in Eq. (3-2) is larger than the right side, and that performance improvement can be expected by splitting the query Q into three parts. Next, as an example of a pattern that degrades performance, let us consider the case where the query Q is split into two parts, q_4 to be executed at A and q_5 to be executed at B, and a management query $2split$ is generated. Assuming that these costs are similar to those in Table 3-1, the left and right sides in Eq. (3-2) are respectively as follows:

$$Exec(A,\ Q) - \sum_{i=4}^{5} Exec\left(E_{x_i},\ q_i\right) = 2.18,$$

$$C(2split) - Init(A,\ Q) + \sum_{i=4}^{5} Init\left(E_{x_i},\ q_i\right) = 3.23.$$

In this splitting pattern, as the left side is smaller than right side, it can be concluded that this splitting degrades the performance of the query Q.

Table 3-1: Costs in a query Q as an example.

Var.	Time (s)	Var.	Time (s)
$C(3split)$	4.13	$Exec(B,\ q_1)$	177.35
$C(2split)$	3.02	$Exec(A,\ q_2)$	2978.14
$Init(A,\ Q)$	11.81	$Exec(B,\ q_3)$	1.97
$Init(B,\ Q)$	0.21	$Exec(A,\ q_4)$	3450.59
$Exec(A,\ Q)$	3454.80	$Exec(B,\ q_5)$	2.03

3.3.4.4 Selection Criteria for QE

In the query splitter, it is important to have selection criteria that determine the QE to be executed by each operator. There are two basic concepts what these criteria contain: (1) availability of execution and (2) performance difference between QEs. Availability of (1) is determined by the type of operator and limitations of resource usage. The

decision based on the type of operator is based on the grammatical compatibility of the target QE. The decision based on limitations of resource usage is based on the number of CPU cores, memory, storage, network, and other resources available to the target QE, and the amount of these resources used by the operator. The performance difference of (2) is based on the difference in quantitative processing performance of the target operator, and the QE with higher performance is selected.

The above criteria should be set in advance in the engine information of the virtual schema. This includes the availabilities of execution, performances, and resource usage limitations for all operators that the proposed system can execute. The performance of operators represents the performance of the QE in processing operators, and is set for all operators for each QE. It is used to determine the difference in performance among multiple QEs, and is set as a constant value so that the superiority or inferiority of performance can be determined among QEs.

3.3.4.5 Selection Algorithm

The procedure for determining the appropriate QEs for all operators of a query plan is shown in Algorithm 3-1. First, it selects appropriate QEs for all operators from a query plan *planTree* that has a set of operators for the query. For each operator, the appropriate QE is evaluated using the function *evaluateEngine*(*op*, *engine*) to obtain the adaptivity. The adaptivity is determined by the algorithm described later. If the adaptivity is 1, the QE engine to be evaluated is appropriate, and its QE is set to *op.engine* of the operator *op*; if the adaptivity is 0, this means there is no performance difference, and the QE most recently selected, *preselectedEngine,* is set to *op.engine* in order to reduce the overhead of switching QEs as much as possible. Once the adaptivity of all QEs for an operator op has been evaluated, the cost information necessary to apply the performance model described in section 3.3.4.3 is calculated using the function *CalcCostCriteria*(). The above is repeated for all operators in the *planTree* to complete the QE selection. Finally, the performance of the split query is evaluated with the function *checkPerformance*(*planTree*) that evaluates Eq. (3-2) using the cost information for each operator. If it is determined that the split query degrades the performance, the splitting is stopped, and the execution is switched to a single QE that can be executed reliably.

Algorithm 3-1: Selecting Query Engine (*selectEngine*)

```
    Input: planTree: Query plan,
           engineInfoList: Query engine list
    Output: planTree: Query plan binding query engine
 1  preselectEngine = null;
 2  for each operator op in post order from planTree
 3  |  op.engine = null;
 4  |  for each engine from engineInfoList
 5  |  |  adaptivity = evaluateEngine(op, engine);
 6  |  |  if adaptivity is 1
 7  |  |  |  op.engine = engine
 8  |  |  else if adaptivity is 0 and
 9  |  |  |       preselectedEngine equals to engine
10  |  |  |  op.engine = engine
11  |  |  |
12  |  |
13  |  If op.engine = null
14  |  |  error("Can't execute this query");
15  |  |
16  |  op.calcCostCriteria();
17  |  preselectedEngine = op.getEngine();
18  |
19  If checkPerformance(planTree) is false
20  |  engine = engineInfoList.getSafeEngine();
21  |  selectEngine(planTree, engine);
22  |
```

The algorithm for *evaluateEngine*(*op*, *engine*), which determines adaptivity, is shown in Algorithm 3-2. As described in 3.3.4.4, the algorithm uses the selection criteria of the QE engine in the engine information of the virtual schema. First, the function executable(*op*) determines whether the target operator *op* is executable in the selected QE or not. If it is executable, the performance of the *op* with a selected QE *op.engine* is compared with the one with the other QE *engine*. This allows the more appropriate QE can be selected if there are more than two executable QEs. The function *engine.getPerf*(*op*) obtains a predefined constant value of the performance of the QE engine in the operator *op*. It returns 1 if the performance is higher or *op.engine* is null, -1 if it cannot be processed or the performance is lower, and 0 if there is no difference.

Algorithm 3-2: Evaluating query engine (*evaluateEngine*)

```
      Input: op: an operator of the original query,
             engine: evaluated query engine,
      Output: adaptivity: result of evaluation
 1    if engine.executable(op) is true
 2    |  if op.engine is null
 3    |  |  adaptivity = 1;
 4    |  else if engine.getPerf(op)
 5    |  |        > op.engine.getPerf(op)
 6    |  |  adaptivity = 1;
 7    |  else if engine.getPerf(op)
 8    |  |        < op.engine.getPerf(op)
 9    |  |  adaptivity = -1;
10    |  else // same performance
11    |  |  appropriateLevel = 0;
12    |  └
13    else
14    |  adaptivity = -1;
15    └
```

3.3.5 System Management

In order to operate the proposed system enabling the query splitting, it is necessary to register the selection criteria for all operators of all QEs to be connected in advance. For this purpose, the system administrator needs to calculate the upper limit of available resources for each QE and the relative performance for each operator as the characteristics of the QEs. In addition, at the time of system installation, it is necessary to reserve storage capacity for QEs for the amount of intermediate data generated by switching QEs. Although this is only a temporary use, if there is no free storage capacity in the QE, the query process will fail because the intermediate data cannot be exchanged between the QEs. Since this capacity varies depending on the query to be executed, it is necessary for the system administrator to know the queries to be submitted in advance and reserve the capacity. Otherwise, the system operates on the assumption that query execution will fail if the capacity exceeds the pre-determined capacity.

3.4 Implementation of the Prototype System

3.4.1 Target systems

The prototype of the prototype system targets the SQL-on-Hadoop and the full-text search engine on Hadoop HDFS as the shared storage. Since these QEs share the same Metastore that holds schema information, they perform query processing on the same data. We use Hive [42] and Impala [43] as SQL-on-Hadoop and Solr [49] as the full-text search engine. Solr's index is placed on Hadoop HDFS using Cloudera Search.

The Hadoop distribution used here is CDH5.0. The prototype system consists of five servers: three slave nodes running Data Node, Task Tracker, and Impalad; one master node running Name Node, Job Tracker, Impala Statestore, and Zookeeper; and one client running Hive Server, Hive Metastore Server, Solr Server and the proposed system. PostgreSQL 8.4.13 is used for Metastore and storing the virtual schema of the proposed system. The OS of all servers is CentOS 6.4. The hardware environment is shown in Table 3-2. These servers are all connected to the same network with 1GbEthernet.

Table 3-2: Hardware environments of our prototype system

	Slave node	Master node	Client
Server	Dell PowerEdge 1950	Sun Fire X2100 M2	Dell PowerEdge 2950
CPU	Xeon 5160 3.0GHz Dual-Core x 2	Dual-Core AMD Opteron 1222 3.0GHz	Xeon E5410 2.33GHz Quad-Core
Memory	4GB x 8	2GB x 4	4GB
HDD	1.5TB	1TB x 2	1TB x 4

3.4.2 Selection Criteria of QE in the Prototype Systems

Hive is designed to perform distributed processing with MapReduce for high-speed execution of large data sets. However, MapReduce itself is designed for batch processing and is not suitable for interactive processing. For examples of this reasons, MapReduce incurs overhead for large-scale data processing, such as initialization for each job and disk usage. Especially for complex queries that include subqueries, multiple jobs are

executed and the above overheads become larger.

On the other hand, Impala does not use MapReduce, which reduces the overhead of Hive and enables faster query processing. However, since all data is always allocated in memory for faster processing, swapping occurs when the size of the table to be processed exceeds the total physical memory available for the entire cluster. This results in a significant degradation of query processing performance. In some cases, the system aborts due to out-of-memory error. Therefore, to make effective use of Impala, the size of the target table and the size of the intermediate data generated by the query process must be less than the total physical memory size.

The interface between Hive and Impala is HiveQL, an SQL-like query language, and some operators are not available for either depending on the version of these implementations. However, this implementation targets only queries that can be executed in both of Hive and Impala, and the proposed system does not split a query based on differences in operators.

Based on the above, the selection criteria for Hive and Impala are the size of the data to be processed. Impala is assigned in the case where the total physical memory size of the entire cluster is not exceeded, and Hive is assigned in the case where the total physical memory size of the entire cluster is exceeded. The engine information in the virtual schema of the prototype contains the memory availability limit and the difference in processing performance as information used in the adaptivity evaluation *evaluateEngine*() of Impala and Hive. For the memory availability limit, Hive is set to have no data size limit, which is always true for *executable*(*op*) in the adaptivity evaluation, while Impala is set to 67.2 GB, which is 70% of the total physical memory size of the three slave nodes. Impala is assumed to be superior to Hive for all operators. Note that the prototype system does not support the UNION operation and the OR operation of the WHERE clause, because Hive does not support the UNION operation. Subqueries in the WHERE clause, which are not supported by Hive, are also not supported.

Since Solr is capable of indexing for full-text search and is superior to Hive and Impala for partial match search on text data, the selection criteria are the performance on text search queries. Therefore, the performance of the LIKE operation in the WHERE clause is set to be superior to that of Hive and Impala. In addition, the other operations in Solr are assumed to be unavailable, and there was no upper limit on the available memory.

3.4.3 Implementation

The prototype query plan is generated using Impala. Specifically, user queries are submitted to Impala with an "explain" clause in order to obtain a query plan. This query plan is lexically analyzed and used as the prototype query plan.

As storage information of the virtual schema, metadata information and statistics of the tables shared by Impala, Hive, and Solr are stored in internal PostgreSQL. And the connection information to each QE is also stored in it as engine information of the virtual schema. A single virtual schema of these multiple QEs on a shared storage for users is realized by providing this storage information to users.

In the prototype system, Hive and Impala are selected by every subqueries for simplicity of the implementation. First, the intermediate data size is estimated for each operator, and the operator with the largest total input data size and intermediate data size is selected for each subquery. This is used as the maximum memory size used in the subquery, and is used to determine the degree of adaptation when selecting QE. The adaptivity evaluation compares the maximum memory size for each subquery with the upper limit of available memory for each QE. If the maximum memory size of the subquery is less than 67.2GB, which is the maximum memory size of Impala, Impala is selected for the subquery based on the difference in processing performance. For Solr, if the WHERE clause contains a LIKE operation, the split query is generated for Solr by extracting only its operator.

The intermediate data size for each operator is estimated from the product of the input data size and the selectivity factor that is the ratio of the intermediate data size to the input data size. For this purpose, the selectivity factor of the operator to be estimated is measured in advance for each table to be processed and each processing condition (join condition of JOIN operator, etc.), and is stored in the storage information. The post-processing attributes are also taken into account, and the intermediate data size is calculated using the ratio of the sizes of attributes that change due to the projection operation.

After query splitting, the split queries are generated as a HiveQL query for Hive and Impala, and a HTTP request URL for Solr. When generating queries across different QEs, HDFS is used for intermediate data transfer between the QEs as a shared data source; since Impala uses Hive's internal cache of metadata, it is necessary to explicitly retrieve metadata updated by Hive. Therefore, when using Impala, the "INVALIDATE

METADATA" query is inserted at the beginning of the split query, and the "REFRESH <table name>" query is inserted after the intermediate data insertion. As an example of generating the split queries, Figure 3-6 shows an original user query, and Figure 3-7 shows the split queries generated from it for Hive and Impala. The leftmost number in Figure 3-7 indicates the order in which the queries are executed, and the brackets next to it are the QEs to be executed. Regarding a case of Solr, since Solr cannot directly insert the query results into HDFS, the query results are written to HDFS and loaded into Hive or Impala using the LOAD clause after the query to Solr is executed.

```
SELECT * FROM a JOIN (
  SELECT id, name FROM b WHERE name = “xxx”
) AS j ON a.id = j.id;
```

Figure 3-6: A sample user query.

```
1(Impala) : INVALIDATE METADATA;
2(Impala) : CREATE TABLE hive_temp (id int, name string);
3(Hive)   : INSERT INTO TABLE hive_temp SELECT id, name
                FROM b WHERE name = “xxx”;
4(Impala) : REFRESH hive_temp;
5(Impala) : SELECT * FROM a JOIN hive_tmp as j on a.id = j.id;
6(Impala) : DROP TABLE hive_tmp;
```

Figure 3-7: Sample split queries generated by the prototype system.

3.5 Evaluation

The proposed system achieves both speed-up and reducing the user's effort by executing queries selected appropriate QEs based on the availability of operators and performance differences. To evaluate the effectiveness of the proposed system, we use the prototype system described in section 3.4, and evaluates query splitting based on the differences in processing size in Section 3.5.1 and on differences in operators in Section 3.5.2. The evaluation of query splitting based on the differences in processing size shows that the proposed system is effective for QE combinations in which resource utilization determines whether operators can be executed or not and processing performance.

Furthermore, by evaluating query splitting based on the differences of operators, we show that the proposed system is effective for a combination of QEs in which the availability and performance of operators differ for specific data and operators.

3.5.1 System evaluation (Hive / Impala)

In this section, we evaluate the functionality, performance, and intermediate data size estimation of the prototype system using Hive and Impala for query splitting based on the differences in processing size. The functional evaluation confirms that the proposed system resolves the three issues of the conventional data virtualization systems. The performance evaluation quantitatively evaluates the performance of the prototype system obtained by solving the above issues. In the intermediate data size estimation, we evaluate the accuracy of the intermediate data size estimation, which is used as the selection criteria in the query splitter.

3.5.1.1 Evaluation Environment

The prototype system described in Section 3.4 is used in this evaluation, but the assignment to Solr is not performed here; the same environment was used for the runtimes of Hive and Impala alone. TPC-H[68] is used as a benchmark for this evaluation, and four Scales that represent table sizes is used: 50 (50 GB total), 100 (100 GB total), 150 (150 GB total), and 200 (200 GB total). However, since TPC-H queries cannot be executed by Hive and Impala without modification, queries corresponding to the HiveQL specification are used in [69]. The two main changes in the benchmark are as follows: (1) Subqueries with EXIST and IN clauses, which Impala does not support, are split into two separate queries. (2) Comma-separated implicit JOINs in the FROM clause were explicitly changed to subquery with JOINs. The used TPC-H queries exclude queries that include UNION and OR operations not yet supported by the prototype system and subqueries in the WHERE clause. Q9 and Q21 were excluded because the processing order of the query plan obtained from Impala and the order of the original subqueries were different, and thus could not be handled by the prototype system specification, which processes subqueries on a per-subquery basis.

When choosing either Hive or Impala without using the prototype system, the user needs to decide which is available in the user query. In this environment, even when running Scale 50, the maximum memory size of the user query may exceed the Impala's

upper limit of available memory, 67.2 GB, when considering JOINs and so on. So, the user cannot judge whether the Impala is available for the query or not. Therefore, we will evaluate the performance of the prototype system by comparing it with Hive, assuming that Hive will be selected because it can be executed surely.

3.5.1.2 Functional Evaluation

Figure 3-8 shows the ranges of the 13 TPC-H queries executed by Hive and Impala through the prototype system in units of subqueries (including the base query), which are the units of splitting, by Scale. This shows that the proposed system is able to automatically select the appropriate QEs transparently to the user. Queries Q2, Q4, Q5, Q8, Q15, Q17, Q18, and Q20 are composed of multiple subqueries or multiple queries. Among the results, in queries Q5, Q8, and Q17, the query splitting is performed by selecting a more appropriate QE in a single query in subquery units. This indicates that the proposed system enables automatic selection of QEs on an operator-by-operator basis.

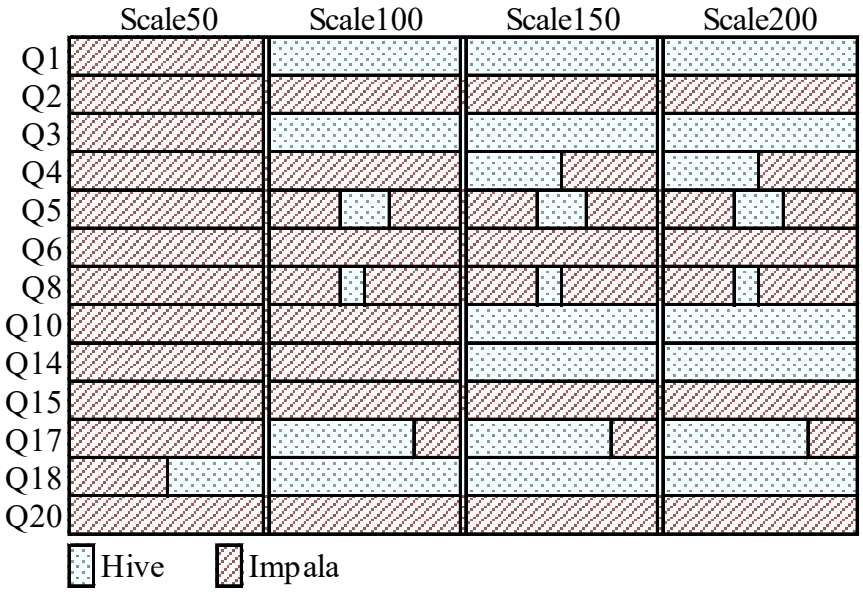


Figure 3-8: Areas of subqueries that QE executes in each TPC-H query.

3.5.1.3 Performance Evaluation

The same 13 TPC-H queries as in section 3.5.1.2 are executed three times each on the

prototype and Hive, and the average execution time is measured. Following four figures show the results, with each graph representing a different Scale of the TPC-H dataset. Figure 3-9 depicts the result of Scale 50, Figure 3-10 depicts the result of Scale 100, Figure 3-11 depicts the result of Scale 150, and Figure 3-12 depicts the result of Scale 200.

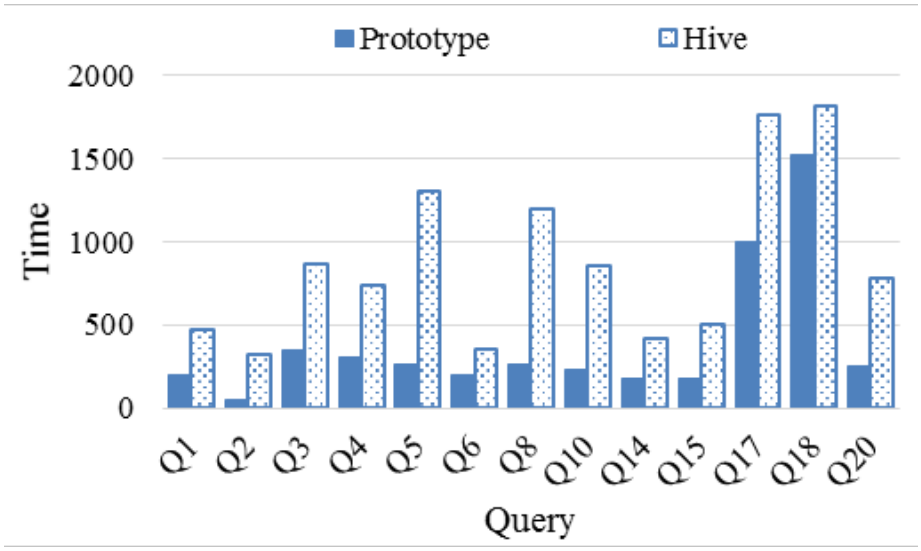


Figure 3-9: Performance evaluation results on SQL-on-Hadoops with Scale 50.

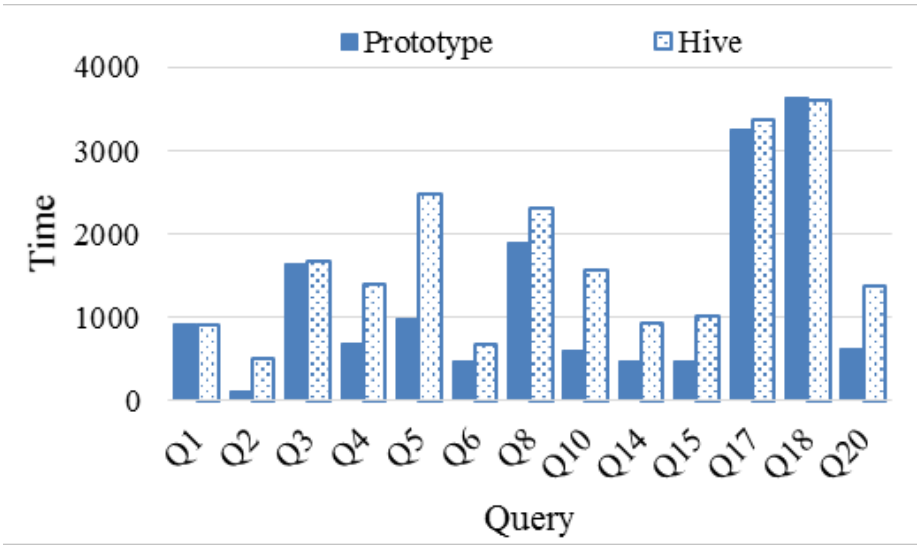


Figure 3-10: Performance evaluation results on SQL-on-Hadoops with Scale 100.

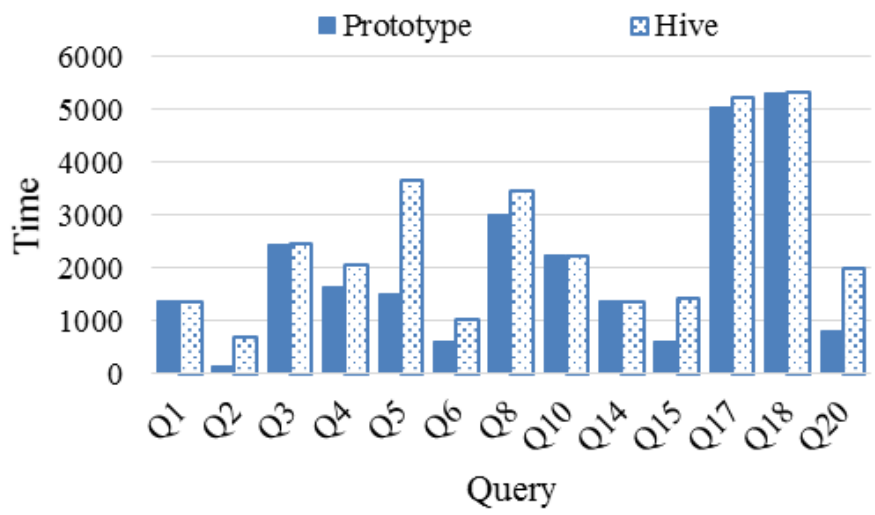


Figure 3-11: Performance evaluation results on SQL-on-Hadoops with Scale 150.

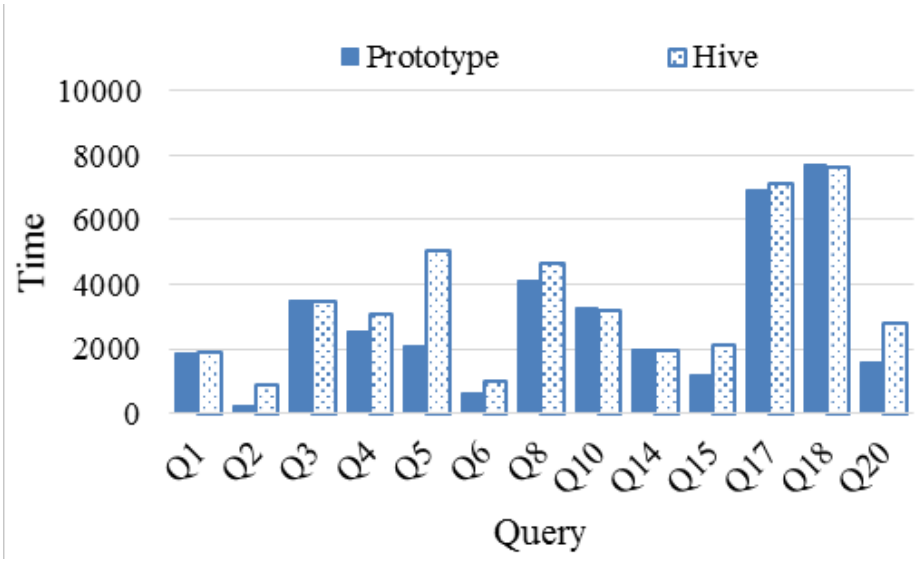
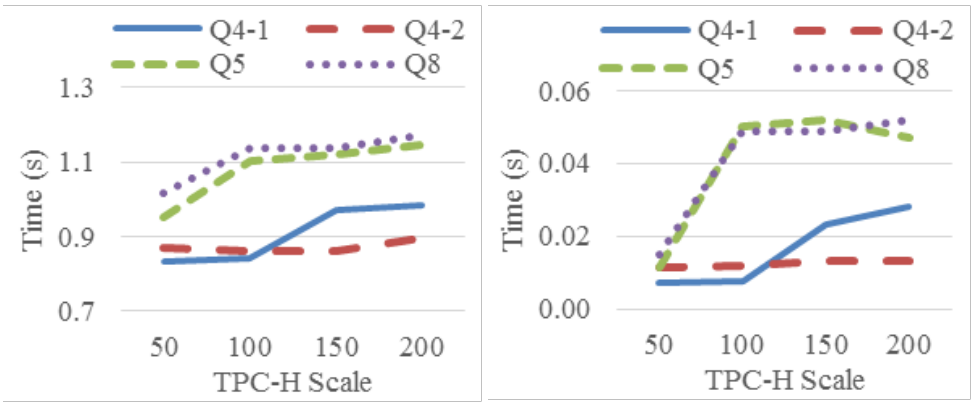


Figure 3-12: Performance evaluation results on SQL-on-Hadoops with Scale 200.

As described in section 3.5.1.2, queries Q1, Q3, Q6, Q10, and Q14 executed in the proposed system are assigned to either Hive or Impala. As a result of the performance evaluation, the queries and Scales assigned to only Hive by the prototype system (e.g., query Q1 in Scale 100) show almost the same results as those of Hive. On the other hand, the queries and Scales assigned to only Impala by the prototype system (e.g., query Q1

in Scale 50) show a significant performance improvement. As for the queries other than the above, the queries and Scales with all subqueries allocated to Impala was faster than Hive. In addition, the queries and Scales with some subqueries allocated to Impala (e.g., query Q18 in Scale 50) was faster than Hive alone for all of them, according to the query splitting method in the proposed system.

Next, we evaluated the proposed system's overhead, which cannot be taken into account in the performance model. In conducting this evaluation, we classified the overhead to be evaluated into two major categories. One is the initialization process, including the process of connecting to the virtual schema and various data sources, and generating split queries. The other is the termination process of disconnecting from the connected virtual schema and various data sources. Among the queries executed in section 3.5.1.2, we selected the characteristic queries Q4, Q5, and Q8 as the queries to be measured. These queries have different processing complexity, with 2, 5, and 8 subqueries including the base query, respectively. Figure 3-13 shows the measured initialization and termination times for these three queries. Query Q4 is divided into queries Q4-1 and Q4-2 in the figure. Query Q4-1 is a subquery in the EXIST clause of query Q4, and query Q4-2 is a JOIN between the result of query Q4-1 and the table that was the target of EXIST processing. query Q4-2 is a JOIN to the table that was the target of the EXIST process.



(a) Initialization (b) Finalization

Figure 3-13 Evaluation Results of the overhead.

For the initialization process in Figure 3-13 (a), the processing time is proportional to the number of subqueries, except for Scale 150 and 200 in Q4-1. The results for the termination process in Figure 3-13 (b) are almost the same, but the time order differs by an order of magnitude, with processing times ranging from 0.01 to 0.05 seconds. These overheads are less than 1% of the query execution time and have no impact on the prototype system.

The above performance evaluation indicates that the proposed system is effective for QE combinations where resource usage of the query determines operator availability and processing performance.

3.5.1.4 Evaluation for Intermediate Data Size Estimation

This section evaluates the accuracy of the prototype's intermediate data size estimation based on the selectivity factor, since it is an important factor in determining the QE to be executed and has a significant impact on the proposed system's problem solving. Here, we measured the maximum memory sizes estimated by the prototype system for each subquery using queries Q4, Q5, Q8, and Q17 of Scale 150, which is partitioned and executed on both Hive and Impala. The results are shown in Figure 3-14. The horizontal axis represents the execution order of the subqueries, and the vertical axis represents the estimated maximum memory size. The unmarked line represents the upper limit of Impala's memory availability, and if it is exceeded, the subquery is executed in Hive.

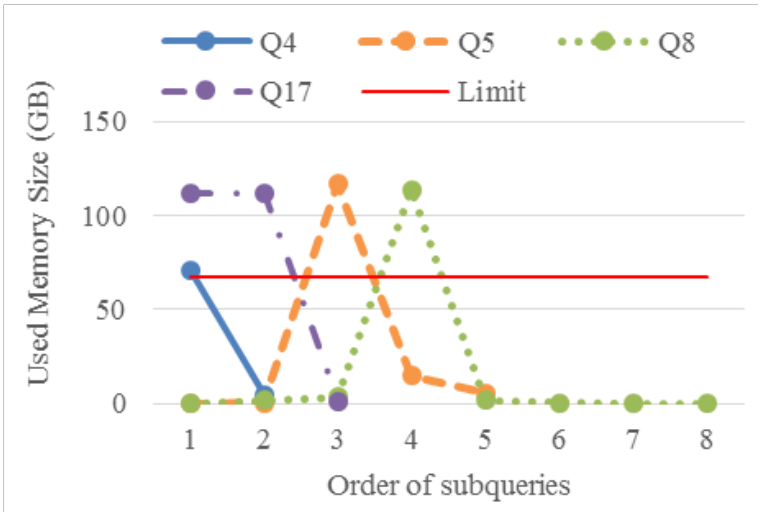


Figure 3-14: Estimated maximum memory sizes of each subquery on scale 150

In addition, we measured the actual intermediate data size for the same query, and calculated the maximum memory size for each subquery. The ratio of the estimated maximum memory size to the actual maximum memory size is shown in Figure 3-15 as the error rate. An error rate of 1 indicates that there is no error, a rate greater than 1 indicates that the maximum memory size is estimated to be larger, and a rate less than 1 indicates that the maximum memory size used is estimated to be smaller. Since the TPC-H dataset does not change characteristics of data component with changes in Scale, the error rate is also not affected by changes in Scale and is the same for all Scales, so only Scale 150 is shown in the figure.

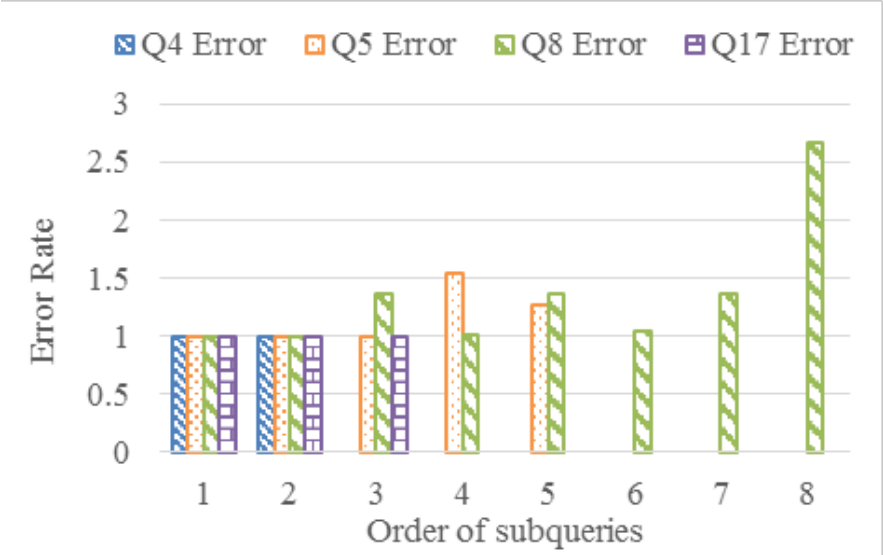


Figure 3-15: Error rates of estimated maximum memory size.

This result shows that the first half of each query is almost error-free, but errors occur in the second half of each query. However, the actual maximum memory size for the subqueries with errors is sufficiently small compared to the upper limit of available memory, indicating that the errors are not large enough to cause errors in the selection of QEs in this evaluation.

3.5.2 System evaluation (SQL-on-Hadoop / Solr)

This section evaluates the performance of the prototype system using SQL-on-Hadoop (Hive or Impala) and the full-text search engine, Solr, with respect to query splitting by

operator differences, and evaluates the effectiveness of the proposed system in QE specialized for specific processing.

3.5.2.1 Evaluation Environment

The prototype system described in section 3.4 is used here as well as section 3.5.1. Note that the query splitting for Hive and Impala is not used here. Three types of data obtained from the Twitter Search API are used for the evaluation: 1-day (about 4.5 GB), 5-days (about 22.5 GB), and 10-days (about 45.3 GB) tweets containing hiragana. The number of records is about 7 million tweets for one day. The tweet data is saved as CSV data in HDFS. Hive and Impala uses these data directly, Solr creates indexes using bi-grams for the text of tweets. The query used for the evaluation extracts users whose tweets contain both “radio” and “bad”, in Japanese “電波” and “悪い”, and outputs the top 10 users with the highest number of tweets. The SQL query for evaluation posted to the prototype is shown in Figure 3-16.

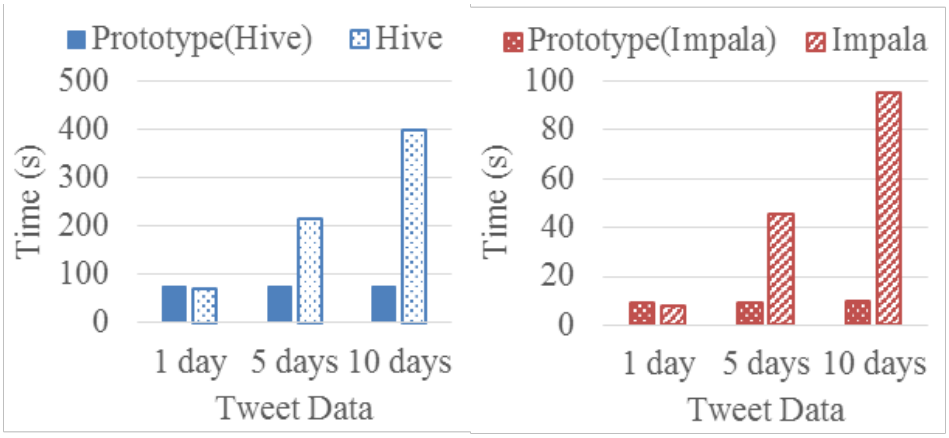
```
SELECT from_user, COUNT(1) AS count FROM tweets
WHERE text LIKE '%電波%' AND text LIKE '%悪い%'
  GROUP BY from_user
ORDER BY count DESC
LIMIT 10;
```

Figure 3-16: Evaluation query for tweet data analysis.

3.5.2.2 Performance Evaluation

Figure 3-17 shows the execution time of the evaluation query, by tweet data size, when the query was executed in SQL-on-Hadoop alone and when the query was split and executed in Solr and SQL-on-Hadoop in the prototype system. Figure 3-17 (a) shows the execution results when Solr and Hive as the SQL-on-Hadoop are used, and Figure 3-17 (b) shows the execution results when Solr and Impala is used. Figure 3-18 shows the split queries generated by the prototype system (excluding management queries), which include Solr query and HiveQL query for SQL-on-Hadoop. The performance of the prototype system is slightly worse than that of Hive and Impala alone for 1-day tweet data. On the other hand, for 5-days and 10-days tweet data, execution time of the prototype system for both Hive and Impala was significantly faster than the stand-alone

execution in proportion to the data size. The prototype system is 3.0x faster for 5-days and 5.3x faster for 10-days than Hive, and 4.8x faster for 5-days and 9.4x faster for 10-days than Impala.



(a) Solr + Hive (b) Solr + Impala

Figure 3-17: Evaluation results for performance on Solr and SQL-on-Hadoop.

```
<Solr>
http://solr-server:8983/solr/tweets_s_shard1_replica1/select?
q=text%3A%22%E9%9B%BB%E6%B3%A2%22+AND+text%3A%22%E
6%82%AA%E3%81%84%22&fl=from_user&wt=csv&csv.header=false

<SQL-on-Hadoop>
SELECT from_user, COUNT(1) AS count FROM tmp1
GROUP BY from_user ORDER BY count DESC LIMIT 10;
```

Figure 3-18: The split queries for Solr and SQL-on-Hadoop, used for the evaluation in Figure 3-17.

The above performance evaluation shows that the proposed system is effective for a combination of QEs with different availability and processing performance for specific data and operators.

3.5.3 Discussion

3.5.3.1 Performance

For some Scale of the TPC-H queries that we evaluated, all subqueries are executed

with Impala, and the results are almost equal to Impala's original performance. To demonstrate this, we measured Impala performance by query Q4. Figure 3-19 shows the results, where the bars indicate the execution times for Impala alone, the prototype system, and Hive alone. As shown in Figure 3-13, there is some delay in the result of the prototype system due to its overhead of query splitting, such as the split query generation, but the difference from Impala's execution time is negligible.

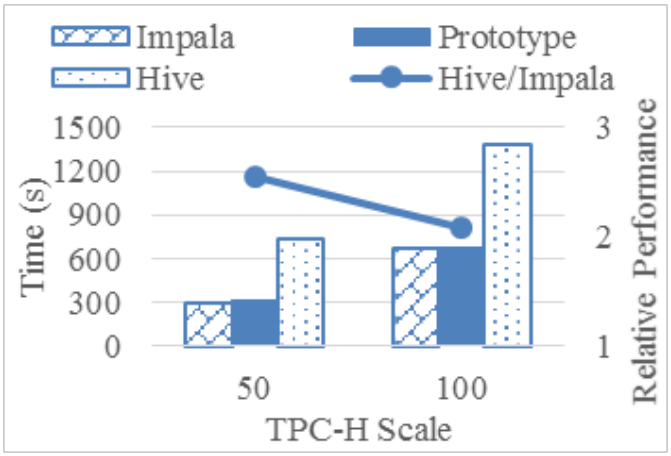


Figure 3-19: Execution time of query Q4 by Impala, the prototype system and Hive, and relative performance of Impala against Hive.

To analyze the performance deference with changes in processing complexity and data size, we consider the performance ratios of the prototype system to Hive using queries Q4, Q5, Q8, and Q17 with Scale 100 or greater, which are split to both Hive and Impala in the prototype system. Figure 3-20 shows how many times faster the prototype system is compared to Hive. Note that these queries exclude the Scale 50 queries performed by Impala. The results show that the combination of Hive and Impala speeds up queries that Impala alone cannot execute up to 2.5 times faster than Hive alone.

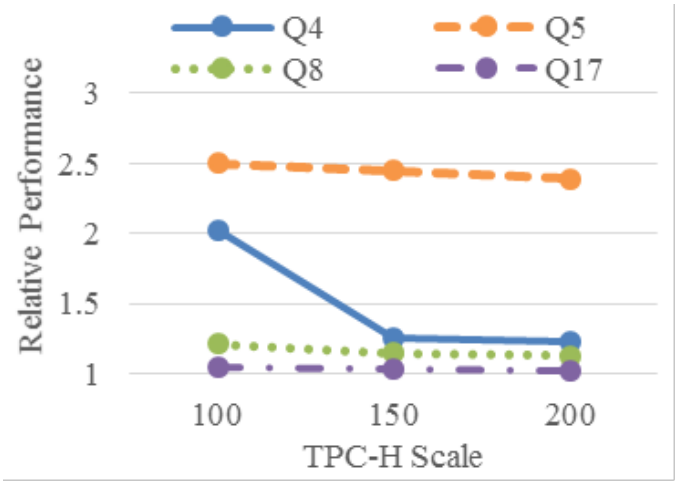


Figure 3-20: Relative performance ratio of our prototype against Hive

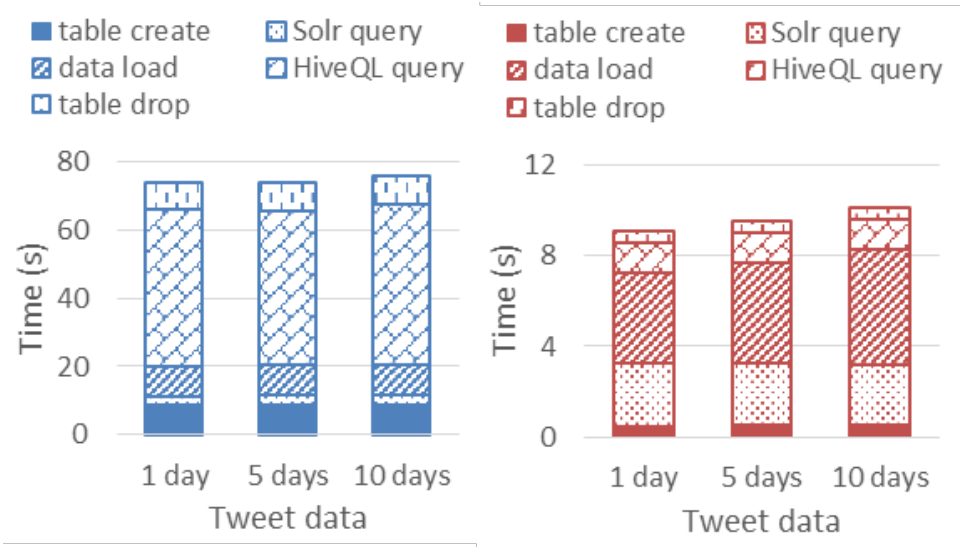
From Figure 3-8, it can be observed that the range of subqueries executed by each QE remains from Scale 150 to 200 for query Q4, and from Scale 100 to 200 for queries Q5, Q8, and Q17. However, in Figure 3-20, the relative performance ratio of these queries tends to decrease with increasing the Scale. A characteristic of the proposed system is that the subqueries are executed by fast QEs, and the performance improvement likewise depends on the performance ratio of these fast QEs and relatively slow QEs. Therefore, the reason for the decrease in the relative performance ratio of the prototype system to Hive despite no change in the split queries is due to the decrease in the relative performance ratio of Impala to Hive with the increase in Scale, as shown in the line graph in Figure 3-19.

The relative performance ratios in Figure 3-20 are larger for queries Q5, Q4, Q8, and Q17, in that order. We analyze the causes of this difference in the performance improvement of the prototype system for different queries. Figure 3-14 shows that the maximum memory utilization of the fourth and fifth subqueries processed by Impala in query Q5 is relatively larger than that processed by the other Impala queries. This is also true for the actual measured maximum memory usage. The ratios of the total intermediate data size processed by Impala to the total intermediate data size of all operators are 11.0% for query Q5, 6.4% for query Q4, and 6.1% for query Q8, nearly twice as large as the ratio for query Q5. The performance improvement for Q4 is also slightly larger than that for Q8, although the difference was smaller. These results

indicate that the size of the data to be processed is also an important factor in determining the performance difference of the proposed system by query splitting compared to the use of a stand-alone QE.

The overhead shown in Figure 3-13 is proportional to the number of connected QEs for both the initialization and termination processes. This can be inferred from the fact that the overhead increases when Scale changes from Scale 50 to 100 in queries Q5 and Q8. This is because the query is split in Scale 100 and executed by both Hive and Impala, while the query in Scale 50 is Impala-only. Similarly, the same behavior was observed in Q4-1 when the query was changed from Scale 100 to 150. However, this is only because the QE, Impala, executing the query is changed to Hive, and this increase is thought to be affected by the change in the time required to establish and terminate connections due to the difference in QEs. There is also a difference in the overhead of each query at Scale 50, where the same number of connections are processed for the same QE, and at Scale 100 to 200 for queries Q5 and Q8. This difference is proportional to the complexity of each query, and is thought to be affected by the increase in the parsing process of the queries and the amount of statistical information to be obtained.

The split queries in SQL-on-Hadoop and Solr showed speedups in both the Hive and Impala cases, with the exception of the 1-day tweet data. To analyze this factor, Figure 3-21 shows the execution times for the split queries including the management queries. Figure 3-21 (a) shows the breakdown of execution time for the split queries for both Solr and Hive, and Figure 3-21 (b) shows the breakdown of execution time for the split queries for both Solr and Impala. Components of the execution time are, from the bottom to the top, temporary table creation, Solr query execution, data loading to the temporary table, HiveQL query execution for Hive or Impala, and temporary table deletion. From these figures, it can be seen that the execution time of Solr is almost constant regardless of the data size to be searched. Therefore, the larger the data size, the faster Solr is than Hive and Impala, which cannot create indexes for full-text search. On the other hand, for a small size such as 1-day, the switching time of QEs is larger than the speed-up by Solr, and the prototype is slower than SQL-on-Hadoop alone. Although we did not perform the performance degradation avoidance here for the sake of evaluation, the performance model described in 3.3.4.3 can avoid such split-query generation.



(a) Solr + Hive (b) Solr + Impala

Figure 3-21: Breakdowns of execution times on the prototype system.

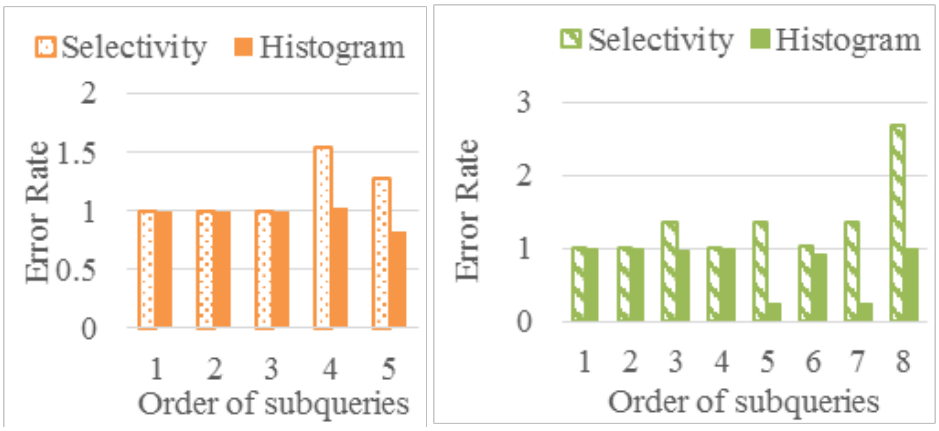
3.5.3.2 Intermediate Data Size Estimation

Since the data size estimation of the prototype system uses the selectivity factor, which is the data size ratio of before and after processing for each operation to a specific data, the accuracy of the estimation does not change as long as the characteristics of the data (variance, etc.) do not change. However, if the characteristics of the data change due to data updates in the same table or due to multiple consecutive processes where the target operator is placed latter, the error becomes larger. Therefore, in Figure 3-15, the error tends to increase in the latter of the process. In queries Q4 and Q17, where the number of subqueries is small, there is almost no error. On the other hand, queries Q5 and Q8, which have many subqueries, have errors up to 2.5 times larger in the latter of the query. This suggests that data size estimation that takes into account the characteristics of the data is necessary when table data is frequently updated or when the data size increases toward the latter of the query.

Therefore, we implemented an intermediate data size estimation method proposed in [70], which uses histograms as a size estimation method that can cope with such changes in data characteristics, and evaluated the estimated data size in the same way. The histogram in this method is statistical information for each attribute. The number of

values for an attribute is aggregated by type of value, called domain. This domain is simplified to a certain range, called bucket, and used as statistical information. This allows us to understand the bias (variance) in the number of domains for each attribute. For size estimation, a histogram is created in advance for all attributes of all tables, and an intermediate histogram for each operator is created at the time of query execution to estimate the memory size used. In this implementation, up to 100 buckets were created with a single attribute. Each bucket was created with an equal number of domains.

Figure 3-22 (a) compares the actual error rates of the selectivity-style method and the histogram-style method for query Q5, and Figure 3-22 (b) compares the error rates for query Q8. Error rate 1.0 in these figures means no error. For query Q5, the selection and merging operations are able to eliminate errors due to changes in data characteristics, resulting in fewer errors overall. For query Q8, the third and eighth subqueries improved, but the fifth and seventh subqueries had large errors, and the memory size is estimated to be small. This is due to a problem with the intermediate data size estimation of the histogram-style for join operation. It tends to estimate the join operation to be small when the aggregated domain in a bucket is not continuous value. Further improvement is needed to achieve more accurate memory size estimation.



(a) query Q5 (b) query Q8

Figure 3-22: Comparisons of error rates of the selectivity-style method and the histogram-style method.

In addition, since errors in intermediate data size estimation are unavoidable, queries

may be re-executed due to errors. A method that avoids this as much as possible is to dynamically determine the QE at runtime [71]. This allows dynamic switching of the QE to be executed while correcting size estimation errors at runtime. Furthermore, this method is also effective when the upper limit of resource availability changes dynamically due to multi-user usage. However, since a split query needs to be generated for each operator and a split query loss occurs, it is necessary to consider the trade-off between the two. Another issue is how to obtain real-time statistics during the execution of the split queries.

3.6 Summary

This chapter describes a proposed data virtualization system that automatically selects an appropriate QE from multiple QEs sharing the same data, and its prototype of the proposed system for SQL-on-Hadoop and full-text search engines. The results of the evaluation with the prototype system using TPC-H benchmarks on Hive and Impala showed that it can execute queries that fail to execute on Impala alone by split queries automatically selecting the appropriate QE on each split query, and that it is up to 2.5 times faster than Hive alone by the query splitting. We also evaluate it for SQL-on-Hadoop and Solr using Twitter data, and confirmed that using Solr for full-text searches can improve the performance of the query processing by up to 9.4 times, and even faster when the data size increases.

The proposed system is expected to be more effective by utilizing various QEs. Therefore, we would like to increase the number of target QEs that the prototype system can handle, and create a model of split criteria that corresponds to the characteristics of more various QEs. In addition, this proposed system deals with only static performance information of QEs as selection criteria in order to select an appropriate QE. However, the performance of QE may vary depending on hardware and user usage of the system. Future issues include the creation of a performance model that incorporates this dynamic performance information and a split-query generation algorithm that selects the appropriate QE based on this information.

Chapter 4
Data Virtualization for Dividing of Large Database System

As a scale of an integrated database system (DB) that has big data increasing day by day such as facility logs about telecommunication infrastructures gets bigger, its resource planning is more complex and its maintenance cost is higher. The virtual database integration technology is now necessary to divide such a big DB to multiple DBs without inconvenience for users. Data virtualization is one of such a technology that integrates only the schemas in multiple DBs and provides the integrated SQL interface for users like a single DB. However, because the data virtualization system accesses data in each DB through network, a communication between a DB and the data virtualization system degrades its query performance as a bottleneck. This chapter describes the functions that data virtualization should provide for the purpose of operating a large database system by partitioning it in order to resolve the bottleneck problem of the data virtualization. This study quantitatively compares and evaluates query processing performance of a data virtualization system implemented with three methods: a query pushdown method, a data placement analysis method, and an intermediate data relocation method.

This chapter is organized as follows: Section 4.1 provides an overview of this study, Section 4.2 describes related studies and their issues, Sections 4.3 through 4.5 describe the methods to be evaluated: query pushdown method, data placement analysis method, and intermediate data relocation method, respectively; Section 4.6 describes the implementation of the three methods, Section 4.7 evaluates them, and Section 4.8 concludes.

4.1 Introduction

4.1.1 Background

Demand for database integration is increasing to utilize various types of data in a unified way, especially in data analysis and machine learning. A general approach of database integration is to physically integrate data distributed in multiple database systems (DBs) into a single DB. However, as a result of the physical integration, the workload on the DBs increases because the number of users of the integrated DBs and their various queries increase. This may make it difficult to design appropriate server resources. In addition, partial failures of the integrated DB affect all users, increasing the maintenance cost of the operation. This problem is noticeable with big DB, especially petabyte-scale. Therefore, there is a need for virtual integration that can divide an existing DB into multiple parts and does not reduce the convenience of existing users. Data virtualization [18] is one method to achieve virtual integration. It has an SQL interface, aggregates only metadata from multiple DBs, and virtually integrates them into a single DB from the user's perspective. This makes it possible to divide a single DB into multiple DBs according to the scale of data to be integrated, the purpose of use, and the frequency of use, thereby minimizing the complexity of resource design and increase in maintenance costs for each divided DB.

The data virtualization system is connected to the physical DBs via a network and processes SQL operations on the system by acquiring data from the physical DBs. Therefore, the communication between the DB and the data virtualization system becomes a bottleneck in the processing of queries for large data or queries that combine multiple data.

The solutions of this communication bottleneck can be categorized into static methods that design data placement at the time of physically dividing the DB and dynamic methods that perform query pushdown and data relocation at the time of query execution. The former static methods include data placement analysis methods that determine the distribution of data to multiple DBs to reduce the amount of communication based on the query logs executed by users in an original single DB [72]. The latter dynamic methods reduce the amount of communication by optimizing the execution location of the computation process of the user's query. There are two methods: (1) query pushdown methods [18][84][85] that reduces the amount of communication data by processing in

a physical DB as much as possible, and (2) intermediate data relocation methods [18] [8][87][88], which temporarily moves the processing result of data located in one physical DB to another physical DB in order to process a query for data that exists in multiple physical DBs.

4.1.2 Contributions

This study quantitatively compares and evaluates the effectiveness of static and dynamic methods that can solve communication bottlenecks in data virtualization in terms of query processing performance. The evaluation uses queries that measure tele-communication quality by using tele-communication infrastructure operation logs as real data. The results of this evaluation show that the communication bottleneck in DB dividing by data virtualization can be largely eliminated by applying three methods: a data placement analysis method, a query pushdown method, and an intermediate data relocation method.

The contributions of this study are as follows:

- The implementation of a data virtualization system that can comprehensively evaluate three methods to solve communication bottlenecks in data virtualization: a data placement analysis method, a query pushdown method, and an intermediate data relocation method.
- The quantitative evaluation of the effectiveness of these methods using real data and queries.

4.2 Related work

Data virtualization is a database integration technology that virtually integrates multiple DBs into a single DB [18]. Unlike database federation [82], in which a DB itself cooperates with other DBs, such as the Foreign Data Wrapper [80] in PostgreSQL and Database Link [81] in Oracle database, data virtualization systems are built completely independent of the integrated DBs. As shown in Figure 4-1, users submit SQL queries to the data virtualization system, and the system retrieves necessary data from the DB and processes the queries.

Data virtualization systems that virtually integrate multiple DBs, also called multi-database systems or heterogeneous integrated database systems, have been studied for a

long time [96]. As projects related to data virtualization systems, GRelC [57], OASIS [58], and ViDa [59] have data access modules, called Wrapper, that support various protocols. In the query processing, data is retrieved through the Wrapper, and all arithmetic operations are performed on the data virtualization system. As a result, communication becomes a bottleneck when data is large.

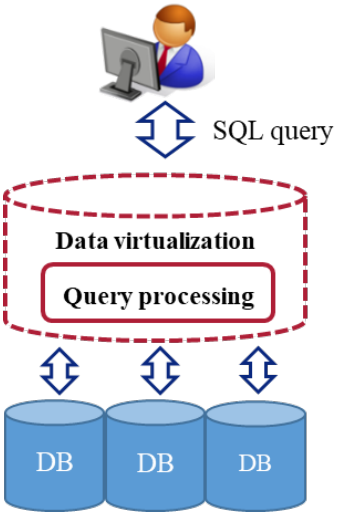


Figure 4-1: Data virtualization System

MIND [86] is one of data virtualization system that adopts a query pushdown method [18][84][85] for RDBMS to delegate as much processing as possible to reduce the communication volume. However, for join operations on data from different DBs, query processing is performed after data transfer to MIND as data virtualization system. This means that MIND does not solve the communication bottleneck for join operations on large data.

The following two types of data placement methods can be applied to query pushdown, so that queries including join operations with large data can be processed in the physical DBs. First, a query log analysis [72] is performed before query execution as a static method. This places tables that are communication bottlenecks in the same DB in advance. Next, as dynamic method, an intermediate data relocation method [18][8][87][88] is proposed to reduce communication cost by dynamically moving intermediate data between DBs during query execution and query pushdown of the join process.

Examples of these data placement methods applied to data virtualization include HP Labs' Pegasus [52], TSIMMIS [53], which also targets unstructured data such as files, IBM's Garlic [54], which also targets multimedia data, and HERMES [51], which aims to construct inference systems. These cases apply intermediate data relocation methods for the purpose of integrating multiple DBs that already exist. However, because they do not assume to divide a DB, static data placement analysis methods based on query log analysis have not yet been applied. Therefore, a comparative evaluation of both static and dynamic data placement methods with query pushdown, and the possibility of resolving communication bottleneck issues when they are used in combination have not been discussed.

In this chapter, we implement a data virtualization system that can evaluate three methods: query pushdown, data placement analysis, and intermediate data relocation, as a virtual integrated environment shown in Figure 4-2, and conduct evaluation experiments using real data. In this evaluation, a communication bottleneck is defined as a delay in the execution time of a query in a virtual integrated environment compared to the execution time of a query in a single DB environment, and the resolution of communication bottleneck issues is evaluated by comparing the execution time delay with and without the three methods.

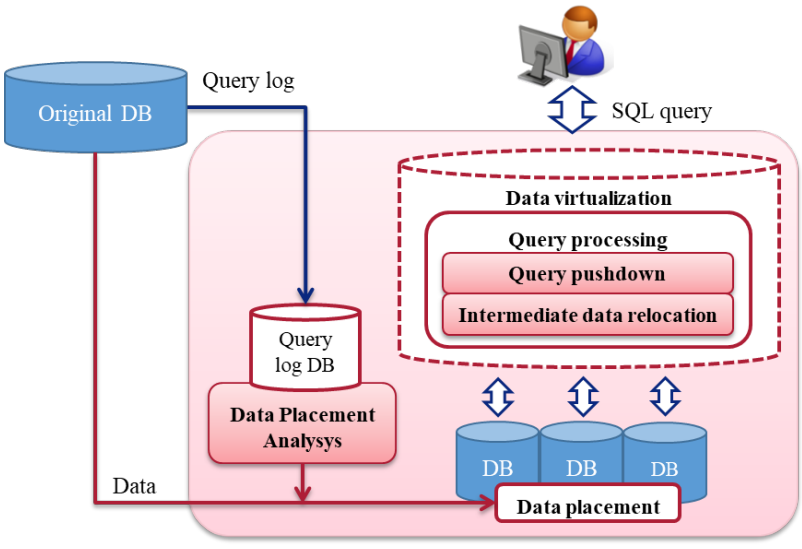


Figure 4-2: A virtual integrated environment using data virtualization.

4.3 Query pushdown method

Many operations in SQL queries have the effect of reducing the data size by processing them, such as selection operation and aggregation operation. In DBMSs, there are rule-based optimization techniques that reduce the size of the target data for subsequent operations by processing selection and projection operations as soon as possible after table loading [83]. In the same way, processing in DB as much as possible after table loading is expected for the data virtualization system to reduce communication volume [18][84][85]. Query pushdown is a technique that takes advantage of this effect by delegating all operations that can be processed in a same DB to the DB. The method is applied to the execution plan after other query optimization methods have been applied, and a sub-tree of the execution plan that can be processed on the same DB is converted into a query to the DB. The data virtualization system processes the remaining operations of the execution plan on the result of this query.

On the other hand, some join operations, such as cross joins and many-to-many joins where one record joins multiple records, increase the data size, so the query pushdown rather increases the communication volume. This increase in communication volume can be avoided by excluding the corresponding operations from the query pushdown. However, by excluding the join operation from the query pushdown, the subsequent operations cannot be query pushed down as well. As a result, the communication volume may increase. To solve this problem, a method has been proposed to create a join graph that represents the join relationship between tables from a query as a graph, and decompose it into subqueries for query pushdown [85]. This not only avoids cross joins that are not represented in the Join Graph, but also reduces the amount of communication by decomposing the edges of many-to-many relationships as well as the edges of different DB tables.

4.4 Data placement analysis method

In order to increase the effect of processing operations in DBs by query pushdown, it is desirable to minimize join operations between DBs as much as possible, and even when join operations between DBs are necessary, the data should be located to minimize communication cost [72]. This can be achieved by analyzing the SQL queries executed

in the original DB before the dividing, and placing the tables that are subject to join processing with particularly large communication costs into the same DB.

In SQL query analysis, various information as features is extracted from SQL queries and their workloads for the purpose of query processing cost estimation, schema mapping, and so on. One of the simplest methods is to extract information by analyzing SQL queries as text data [73][74]. There is also a method to analyze queries by creating a Join Graph from SQL queries, where nodes are tables and edges are join relations [75][76]. On the other hand, the processes expressed in the SQL query statement may differ from the actual processes due to query optimization by the DB and other factors. Furthermore, because the above methods cannot take into account the ordering of join operations, they do not take into account information about the actual processing. As a query analysis method that takes these factors into account, there is a method for analyzing query execution plans [77].

4.5 Intermediate data relocation method

After generating the subqueries for the query push down, it is necessary to determine how to combine the results of each subquery. In MIND [86], one of data virtualization system, the subquery results are acquired and processed by the data virtualization system. When a subquery result is large, communication by its data acquisitions becomes a bottleneck. On the other hand, by combining the results of two subqueries processed in different DBs, it may be possible to reduce the data size compared to the results of each subquery. Intermediate data relocation is a method to move the result of a subquery, which is the intermediate result of a query, to the DB where the result of another subquery is located and perform the join operation. This technique is applied when communication cost is expected to be reduced compared to the case where the data virtualization system acquires the intermediate data and joins them.

Figure 4-3 illustrates an example of intermediate data relocation for a query pushdown subquery. It is assumed that the result data size of subquery SubQ1 is large, while the results of the other two subqueries, SubQ2 and SubQ3, are relatively small. In this case, when processing in a data virtualization system, the communication of SubQ1 becomes a bottleneck. Therefore, the results of SubQ2 and SubQ3 are relocated to the DB that executes SubQ1, where they are combined to reduce the amount of communication data.

Two major methods have been proposed for the intermediate data relocation method in order to combines the results of subqueries executed in different DBs [18][8]. Ship Join moves the smaller subquery result R to the DB where the other subquery result S is located and performs the join process. In this case, the DB with S requires the creation of a temporary table to store R, the insertion of R, and the deletion of the temporary table after the join process of R and S as additional processes. Another method, 2-way Semijoin, has been proposed. In this method, only the attributes used as join keys from the smaller subquery result R are transferred to the DB where the other subquery result S is located, and Semijoin is performed. The 2-way Semijoin can reduce the amount of communication compared to Ship join when the number of attributes in R is large and when the size of R and S after Semijoin is small. On the other hand, 2-way Semijoin degrades performance when the number of records in R and the number of records in the result of the join are large. To solve this problem, some methods [8][87][88] have been proposed to encode and transfer such as Bloomjoin.

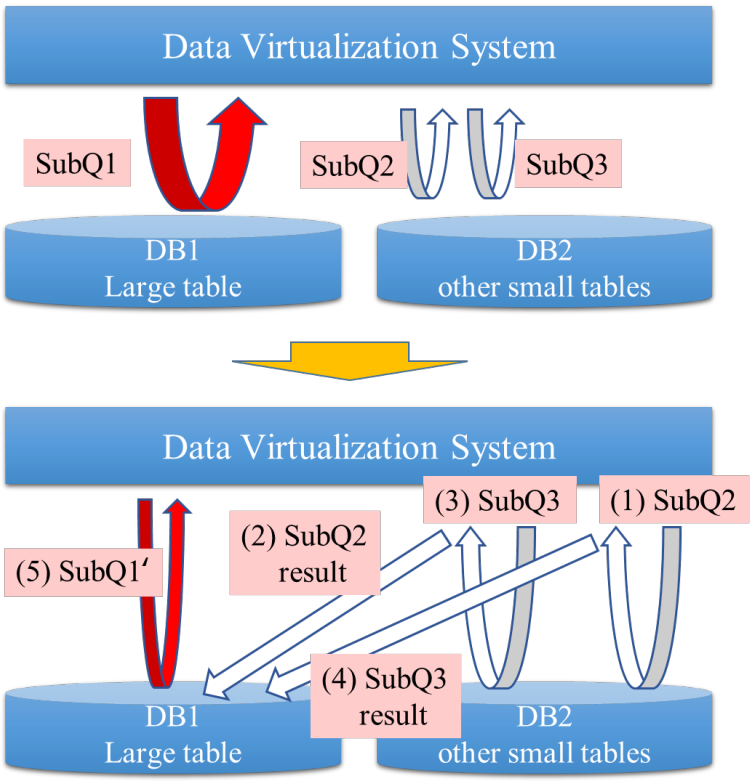


Figure 4-3: An example of the intermediate data relocation.

When there are multiple subquery results to be required join operations, it is necessary to determine the location and order of each join operation. The join order has long been considered in conventional DBMSs. It can be mainly classified into Dynamic Programming (DP), which performs comprehensive search, Greedy Algorithm (Greedy), which performs heuristic search, and Randomized Algorithm, which randomly changes the order of table joins to search [89]. In the case of data virtualization and distributed DBs, it is necessary to consider the execution location in addition to the execution order. Therefore, DP is not realistic from the viewpoint of computational complexity due to its wide search range. Various extensions have been proposed to reduce the computational complexity and get closer to the optimal solution [90][91], such as Iterative Dynamic Programing (IDP) [89], which combines DP and Greedy, and DPccp [92], which executes DP based on Join Graph.

4.6 Implementation

In order to compare and evaluate three methods of data virtualization: query pushdown, data placement analysis, and intermediate data relocation to resolve communication bottlenecks, we implemented the virtual integrated environment shown in Figure 4-2. In the following, we describe how to implement the query log DB necessary for data placement analysis, how to implement data placement analysis, and how to implement the data virtualization system.

4.6.1 Query Log DB

Since the intermediate result sizes of the actual processing during a query is important in data placement analysis, the method of analyzing query execution plans [77] can be leveraged to estimate the intermediate result sizes. The execution plan of an SQL query is represented by a tree structure. To simplify the accumulation and analysis of the execution plan, the query log DB in this implementation leverages the Nested Set Model [78], which represents a query execution plan with a tree structure in a tabular form. In the Nested Set Model, each node representing an operation in the query execution plan is assigned its position information in the tree structure in addition to the operation information. The position information of the tree structure is represented by two numbers. The tree is traversed from the root node in the order of outgoing and returning, and a

number is assigned when the node is reached, thereby expressing the position of the nodes in the tree of the query execution plan.

Below is s an example of Nested Set Model construction in Q3 of TPC-H [68]. Figure 4-4 shows a query execution plan log in Vertica [79]. Vertica stores one line of text from this query log as one record in a system table. The text of this query log is analyzed on a query-by-query basis to construct the tree of operations shown in Figure 4-5. Following the arrows shown in the tree, numbers are assigned by traversing in order from the root node, the SORT operation, and two numbers are assigned to each operation node. Figure 4-6 shows a tabular representation of the numbers assigned to each operation, with column 'lft' as the number assigned in the outgoing traversing and column 'rgt' as the number assigned in the returning traversing.

```
+-SORT [Cost: 672K, Rows: 4M] (PATH ID: 1)
|  Order: sum((lineitem.L_EXTENDEDPRICE * (1 - lineitem.L_DISCOUNT))) DESC, orders.O_ORDERDATE ASC
| +---> GROUPBY HASH (LOCAL RESEGMENT GROUPS) [Cost: 575K, Rows: 4M] (PATH ID: 2)
| |       Aggregates: sum((lineitem.L_EXTENDEDPRICE * (1 - lineitem.L_DISCOUNT)))
| |       Group By: lineitem.L_ORDERKEY, orders.O_ORDERDATE, orders.O_SHIPPRIORITY
| |       Partially sorted keys: 1
| | +---> JOIN MERGEJOIN(inputs presorted) [Cost: 562K, Rows: 4M] (PATH ID: 3)
| | |       Join Cond: (lineitem.L_ORDERKEY = orders.O_ORDERKEY)
| | |       Materialize at Input: orders.O_ORDERKEY, orders.O_CUSTKEY, orders.O_ORDERDATE, orders.O_SHIPPRIORITY
| | |       Materialize at Output: lineitem.L_EXTENDEDPRICE, lineitem.L_DISCOUNT
| | | +-- Outer -> STORAGE ACCESS for lineitem [Cost: 282K, Rows: 33M] (PATH ID: 4)
| | | |       Projection: public.LINEITEM_super
| | | |       Materialize: lineitem.L_ORDERKEY
| | | |       Filter: (lineitem.L_SHIPDATE > '1995-03-12'::date)
| | | |       Runtime Filter: (SIP1(MergeJoin): lineitem.L_ORDERKEY)
| | | +-- Inner -> JOIN HASH [Cost: 82K, Rows: 2M] (PATH ID: 5)
| | | |       Join Cond: (customer.C_CUSTKEY = orders.O_CUSTKEY)
| | | | +-- Outer -> STORAGE ACCESS for orders [Cost: 70K, Rows: 7M] (PATH ID: 6)
| | | | |       Projection: public.ORDERS_super
| | | | |       Materialize: orders.O_CUSTKEY
| | | | |       Filter: (orders.O_ORDERDATE < '1995-03-12'::date)
| | | | |       Runtime Filter: (SIP2(HashJoin): orders.O_CUSTKEY)
| | | | +-- Inner -> STORAGE ACCESS for customer [Cost: 7K, Rows: 387K] (PATH ID: 7)
| | | | |       Projection: public.CUSTOMER_super
| | | | |       Materialize: customer.C_CUSTKEY
| | | | |       Filter: (customer.C_MKTSEGMENT = 'HOUSEHOLD')
```

Figure 4-4: An example of a query execution plan log in Vertica.

query_id	path_id	node	lft	rgt	cost	rows	table_name
Q03	1	SORT	1	14	672000	4000000	
Q03	2	GROUPBY HASH	2	13	575000	4000000	
Q03	3	JOIN MERGEJOIN	3	12	565000	4000000	
Q03	4	STORAGE ACCESS	4	5	282000	33000000	lineitem
Q03	5	JOIN HASH	6	11	82000	2000000	
Q03	6	STORAGE ACCESS	7	8	70000	7000000	orders
Q03	7	STORAGE ACCESS	9	10	7000	38000	customer

Figure 4-5: Query execution plan of Figure 4-4, which is represented by Nested Set Model.

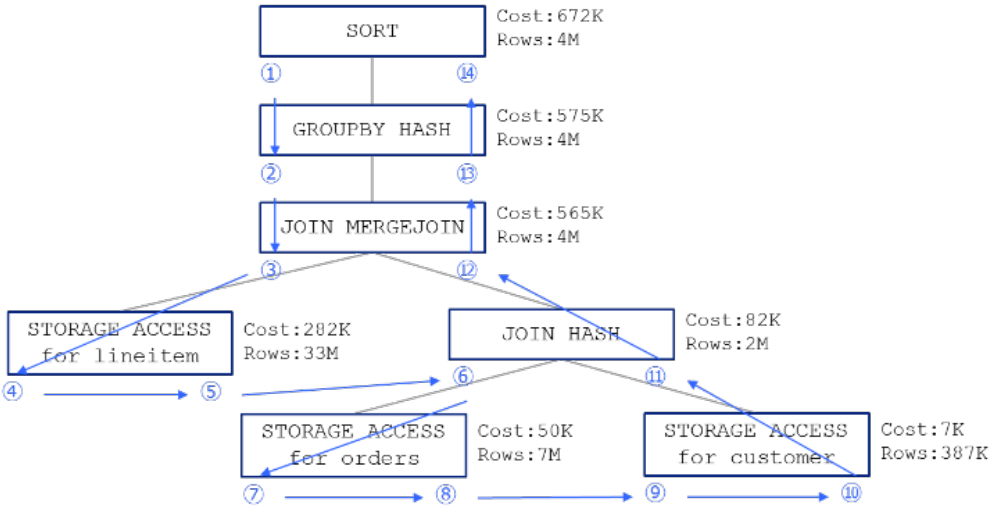


Figure 4-6: Tree structure of Figure 4-5 and position assignment procedure.

4.6.2 Data Placement Analysis Tool

The data placement analysis tool analyzes data placement in divided multiple DBs by utilizing data in the query log DB represented by the Nested Set Model. The inputs for the tool are the execution plans of past SQL queries executed in the original DB, a list of tables to be allocated, and information of target DBs. The output of the tool is the destination DB for each table. Note that duplication and fragmentation of tables are not considered here from the viewpoint of improving maintainability, which is the purpose of DB dividing, because they are assumed to cause increased complexity in maintenance by maintaining data consistency.

The condition for determining the placement of each table is to minimize communication cost in the environment after dividing the original DB. Here, the tool reduce communication for tables joined in different DBs. Therefore, the objective is to collect tables having a join relationship into the same DB. On the other hand, due to the storage capacity limit of each DB as a constraint, it is not always possible to store all the tables having a join relationship in a single DB. It is necessary to separate the joined tables so that the capacity of the DB is less than or equal to the capacity of the DB. Therefore, among the tables having a join relationship, the tables with the highest frequency of their join operations are stored in the same DB. And when the storage capacity limit is reached, the remaining tables are stored in a different DB. The join

relation of the tables is created from the query execution plan in the query log DB. In the execution plan, a join relation is defined as a group of tables required for the join operation, and the frequency of the join operation is summed up. Because joining small tables such as the master table has little impact even if they are joined in different DBs, the pairs of tables that may become communication bottlenecks between DBs are preferentially placed in the same DB by setting the input size and the number of records as conditions for the join relation.

4.6.3 Data Virtualization System

For the implementation of the data virtualization system, we used the open-source software Presto [45], an in-memory distributed SQL query engine that can retrieve and process data from Hadoop and various DBs. We used version 0.152. In this version, the query pushdown can be applied to only projection operations and some selective operations (numerical type comparison operations) by default, while other operations are always executed in Presto. Therefore, we implemented two query processing methods of the data virtualization, the query pushdown and the intermediate data relocation. For the intermediate data relocation, we implemented a histogram-based intermediate result size estimation function [93] to automatically determine whether the relocation process needs to be applied to each join operation, and a function to generate a histogram of a table in the target DB as statistical information for this purpose. Only Ship Join was used as the intermediate data relocation method, and the algorithm for determining the join order and location was based on Greedy to reduce the search time.

4.7 Evaluation

This section evaluates the query pushdown, the data placement analysis, and the intermediate data relocation as three methods to solve communication bottlenecks in data virtualization in a composite manner using facility log data of tele-communication infrastructure and analytical queries. These evaluations measure the execution time of queries and the delay in an environment where a single original DB is divided into three DBs and integrated them virtually using data virtualization. This delay is caused by communication bottlenecks, and the state in which the communication bottlenecks in data virtualization are resolved is defined as the state in which these delays are eliminated.

The following sections first describes the system configuration, data placement, and evaluation queries as the evaluation environment. After that, the results of the evaluation in this environment are described.

4.7.1 Evaluation Environment

Figure 4-7 shows the configuration of this evaluation. The hardware was composed of four servers connected by a 1Gbit Ethernet network. All servers are Dell R720 with 2 Xeon E5-2620 CPUs (12 cores in total), 384GB memory, and 3 HDD 4TB x 3 (RAID10). The OS in these servers is CentOS 6.6. Vertica 7.2.3 [79] is installed on each of the three DB servers. The single original DB (DB-Org) before dividing is built only on Vertica on DB server 1, and the DBs after dividing are built across the three Vertica servers. In addition, two DB environments are built due to the difference in the data placed in each DB server: three DBs divided by data type (3DB-DT) and three DBs divided by data placement analysis (DB-PA). As a result, three logical DBs are built on DB server 1, and two logical DBs on DB servers 2 and 3. Note that the Vertica on each server is not set as a distributed DB. This means data in a different Vertica server cannot be accessed between DB servers.

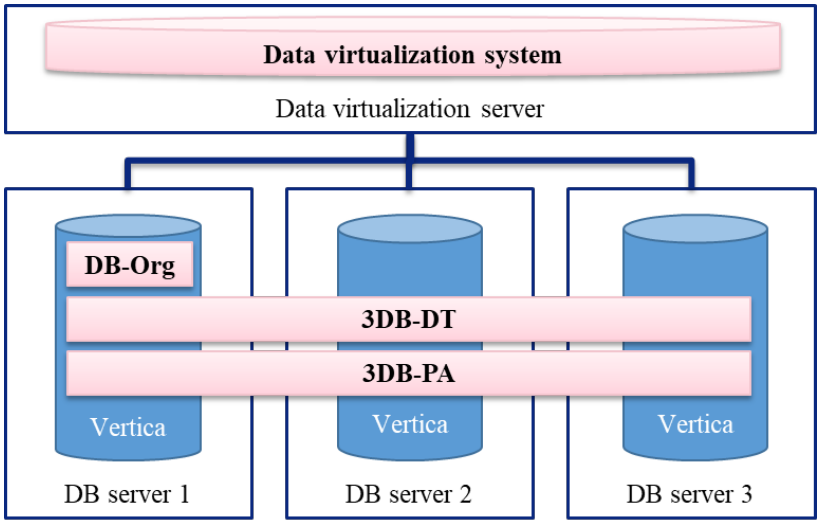


Figure 4-7: Evaluation environment for virtually integrating three DBs.

The evaluation uses 32 tables of facility logs and base station information related to the tele-communication infrastructure, totaling approximately 1 TB of data. Table 4-1

shows the number of tables placed by data type and total data size in the three DB environments. In the 3DB-DT the largest logs (3 tables, 5.9 billion records on average, 79 columns on average, and 102 GB on average), which are fact tables, are placed on DB server 1, and the other fact tables, such as logs of backbone networks (4 tables, 800 million records on average, 81 columns on average, and 1 TB on average), are placed on DB server 2. The master tables regarding base station (25 tables, 500 million records on average, 27 columns on average, 15 GB on average) on DB server 3. The fact tables except for two tables are not joined with each other but with the master tables. The master tables are joined to the fact tables and other master tables using ID keys.

Table 4-1: Data placement by data type and data size in each DB.

Data placement		Number of Tables			Data size
		Fact tables: facility logs	Fact tables: others	Master tables	
DB-Org	**DB server 1**	3	4	25	957 GB
3DB-DT	**DB server 1**	3	0	0	306 GB
	DB server 2	0	4	0	276 GB
	DB server 3	0	0	25	375 GB
3DB-PA	**DB server 1**	3	0	2	348 GB
	DB server 2	0	4	4	344 GB
	DB server 3	0	0	19	266 GB

For the 3DB-PA, the storage capacity of each DB is assumed to be 350 GB, which is about 1/3 of a single DB server. In order to determine the data placement of 3DB-PA, the data placement analysis is performed for 377 ad hoc queries used for tele-communication quality management for one day. The query logs of the above queries executed in Vertica were transformed to the Nested Set Model in advance and stored in the query log DB. The tables with more than 10 million records were clustered by the data placement analysis to be placed in a single DB for join operations between these tables. As a result, the fact tables are same to DB-DT, but two master tables (average 1.7 billion records, average 15 columns, average 21 GB) are placed on the DB server 1, four master tables (average 1.4 billion records, average 36 columns, average 18 GB size) are placed on the DB server 2, and others are placed on DB server 3.

Table 4-2: Information on 12 evaluation queries: type and number of tables to be used, join process across DBs by data placement, query characteristics (purpose, processing details, number of results)

	Number of tables to be used			Scale of join across DBs		Query characteristics		
Query	**Fact tables: facility logs**	**Fact tables: others**	**Master tables**	**3DB-DT**	**3DB-PA**	**Purposes**	**Processes**	**Results**
1	0	0	4	None	None	Aggregation	Union and join between master tables.	2453
2	0	0	4	None	None	Aggregation	Union and join between master tables.	7
3	0	0	2	None	None	Data transform	Join between master tables.	0
4	0	2	0	None	None	Aggregation	Join between fact tables.	3499
5	1	0	3	Small vs. small	Small vs. small	Aggregation	Star join master tables to a fact table.	3
6	1	0	1	Large vs. large	None	Data transform	Join a fact table and a master table.	1818
7	1	0	1	Large vs. large	None	Data transform	Join a fact table and a master table.	2842
8	0	1	1	Large vs. large	Large vs. large	Aggregation	Join a fact table and a master table.	4903
9	1	0	4	Large vs. large	Small vs. small	Aggregation	Join two aggregation results.	74
10	0	1	4	Large vs. large	Large vs. small	Aggregation	Join two aggregation results.	18761
11	1	0	4	Large vs. large	Small vs. small	Aggregation	Join two aggregation results.	0
12	0	0	4	None	None	Data transform	Join between master tables.	343694

The queries used in this evaluation are extracted from the 377 queries used in the data placement analysis, and 12 queries that included large-scale data join processing are selected for this evaluation. Table 4-2 shows information on each of the 12 queries, including the type and number of tables used by the query, the scale of join processing across DBs included in the query, and the characteristics of the query. In the scale of join processing across DBs, the processed data size which is 10 million records or more is defined as large, otherwise as small. "Small vs. small" indicates the joining across DBs of small data, "large vs. small" indicates the joining across DBs of data where one of the DBs is large, and "large vs. large" indicates the joining across DBs of large data. The query characteristics include the purposes, processing contents, and number of results for each query. Each query is classified into two purposes: aggregation of data and data transformation to create intermediate data inserted into a temporary table for use in another query. Note that no insertions into temporary tables were performed during the evaluation. The processing details indicate how to process the data in the query focusing on the join operation.

4.7.2 Summary of Evaluation Results

Figure 4-8 shows the evaluation results of 12 queries executed with and without communication bottlenecks. The vertical axis is the ratio of the query execution time on each data virtualization system to the execution time when the query is executed directly on the DB-Org without the data virtualization system. If this value is 1, the query execution time is the same as that for the DB-Org, and the communication bottleneck is completely eliminated. The vertical axis is displayed in logarithmic scale. Presto with the three methods: query pushdown, data placement analysis, and intermediate data relocation is used as the case where the communication bottleneck is taken into account, while a conventional data virtualization system, original Presto without the three methods, is used as the case where the communication bottleneck is not taken into account. The Presto with the three methods is executed on 3DB-PA as a result of applying the data placement analysis method, and the original Presto is executed on 3DB-DT. Figure 4-8 indicates that the use of the three techniques improves the performance of all 12 queries. The three techniques improve the performance by a maximum delay of 3.8 times (query 5) and an average delay of 1.8 times in the 11 queries except for query 8, while the conventional data virtualization system have a maximum delay of 15,992 times

(query 5) and an average delay of 2010 times in the 11 queries except for query 8.

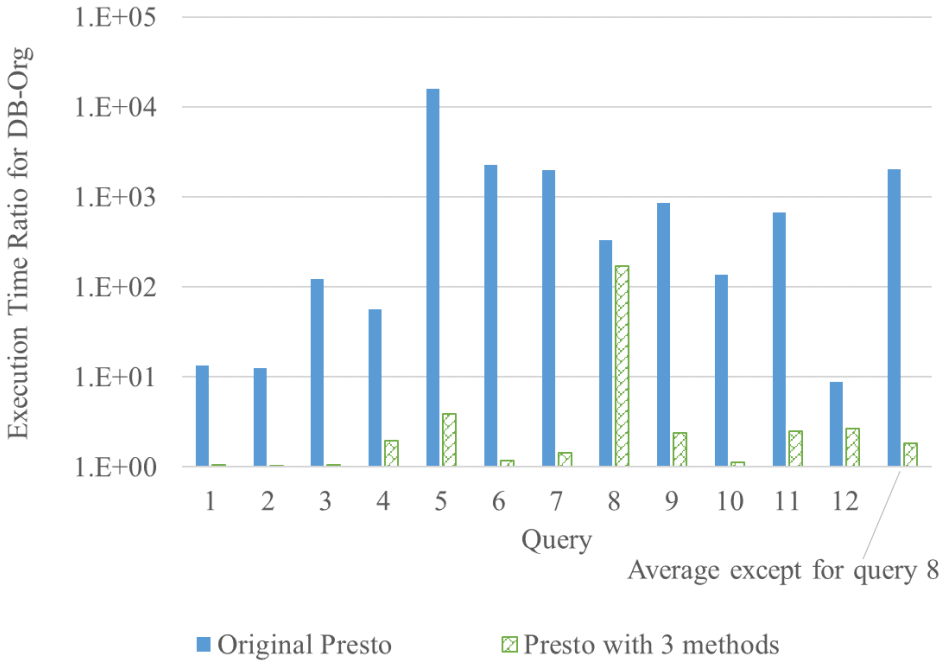


Figure 4-8: Evaluation results of the virtual integration environment using Presto with three methods: query pushdown, data placement analysis, and intermediate data relocation.

4.7.3 Details of Evaluation Results

This section evaluates the effectiveness of each of the three methods in the results of Section 4.7.2.

Because the effect of data placement analysis and intermediate data relocation are based on the implementation of query pushdown, so we first evaluate the effect of query pushdown. Figure 4-9 shows the results of 12 queries executed in the 3DB-DT to which data placement analysis is not applied, with the original Presto and Presto with only the query pushdown. The results indicates that the query pushdown is effective for all 12 queries. On the other hand, queries 6 through 11 are less effective than the other queries, with significant delays ranging from about 60 to 180 times.

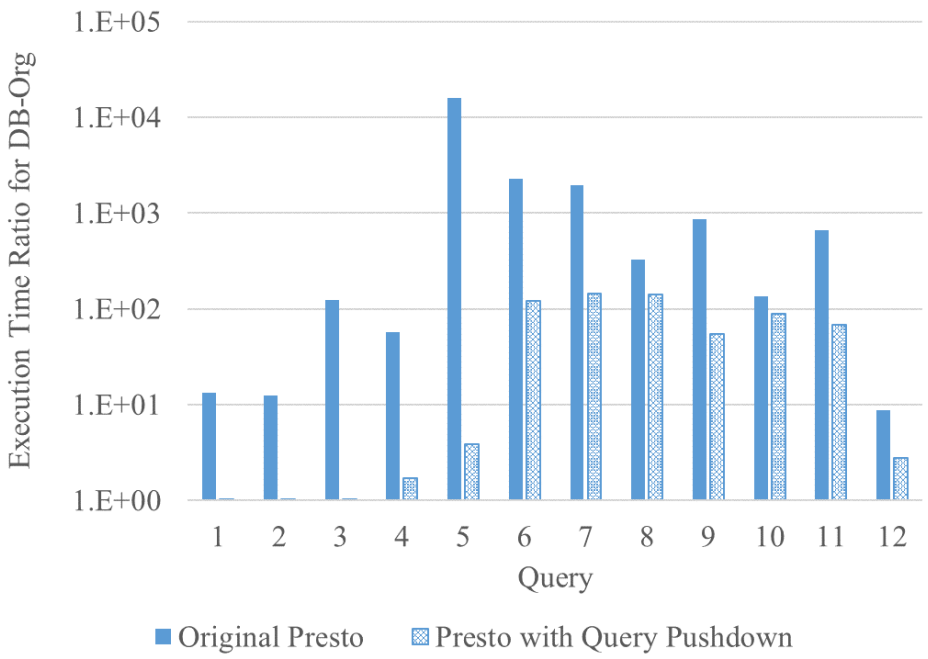


Figure 4-9: Evaluation results of query pushdown in 3DB-DT

Next, we evaluate the effect of data placement analysis with the query pushdown function enabled. Figure 4-10 shows the results of executing 12 queries on 3DB-DT with only the query pushdown, and on 3DB-PA, which is the result of the data placement analysis, with query pushdown. The vertical axis is the execution time ratio as in Figure 4-10, but not in logarithmic form. Four queries showed improved performance due to the effect of data placement analysis, but two queries, 8 and 10, show significant delays as in the 3DB-DT.

Finally, we evaluate the intermediate data relocation of data virtualization. Figure 4-11 shows the results of executing the data virtualization system with only the query pushdown function and, in addition, the intermediate data relocation enabled on the 3DB-PA, which is the result of the data placement analysis. As in the evaluation of the data placement analysis, the vertical axis represents the execution time ratio for the DB-Org. Among the two queries for which significant delays cannot be avoided by the data placement analysis, the intermediate data relocation avoided the delays for query 10. On the other hand, for query 8, the intermediate data relocation works, but the delay is even larger than when it does not work.

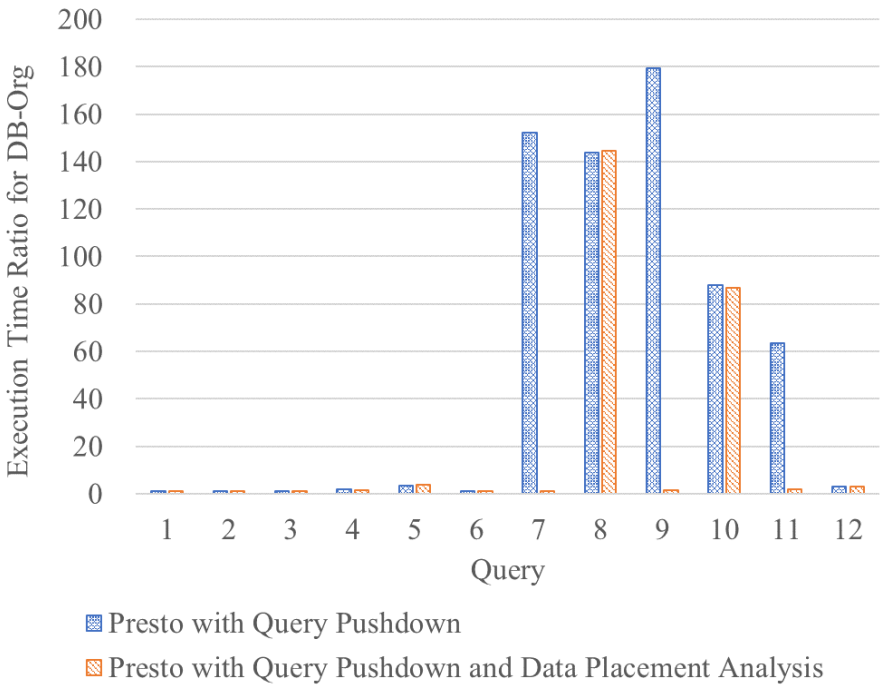


Figure 4-10: Evaluation results of data placement analysis by comparing 3DB-DT and 3DB-PA.

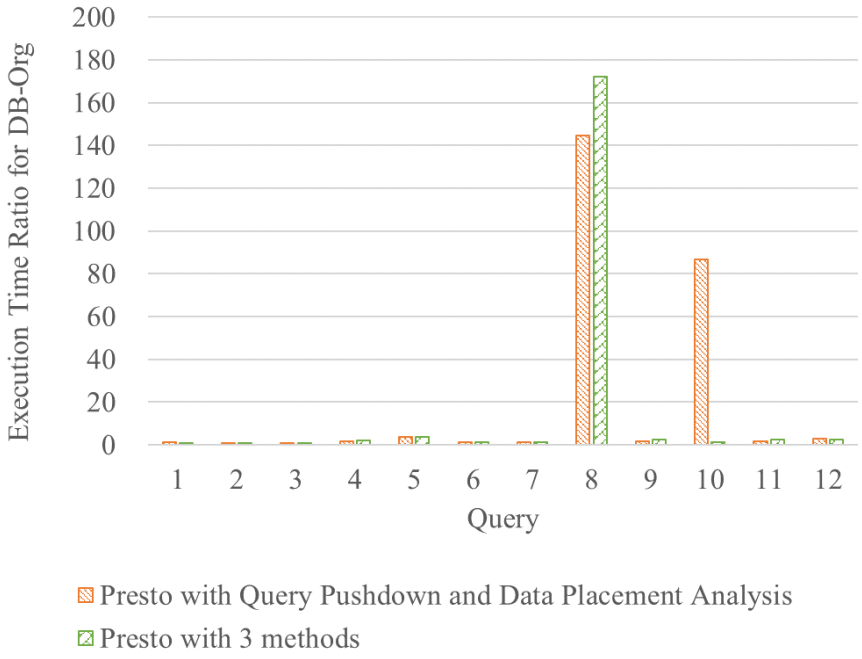


Figure 4-11: Evaluation results of intermediate data relocation on 3DB-PA.

4.7.4 Discussion

4.7.4.1 Discussion for each method

The evaluation results show that a single DB can be split into three DBs without significant delay for 11 of the 12 queries in the real environment by using three techniques: data placement analysis, query pushdown and intermediate data relocation in data virtualization.

The effect of query pushdown is particularly large, contributing to the elimination of communication bottlenecks for all queries. Queries that are completed in a single DB produce results of at most 340,000 records, which is the amount of communication in the query pushdown. Because a table with hundreds of millions of records is communicated when all methods are disabled, the query pushdown greatly reduced the communication cost. For queries that use data in multiple DBs, the amount of communication is also reduced by processing the data in each DB. However, as shown in Table 4-2, queries 6 through 11, which involve the communication of large data with more than 10 million records, still have large delays of more than 10 times. Therefore, unless the results processed in each DB by the query pushdown become smaller, the improvement by the query pushdown alone is limited.

In addition to query pushdown, significant improvements are observed in queries 6, 7, 9, and 11 by improving data placement through data placement analysis. This improvement is due to the fact that the master data moved to DB server 1 enabled query pushdown for large join operations with more than 10 million records. On the other hand, two queries, queries 8 and 10, show no performance improvement. Query 8 do not improve due to the fact that the data placement remained the same. Query 10, which involved moving the master data to DB server 2, makes the result records from the DB server 2 smaller by the query pushdown, but the result of the other DB is still dominant and causes delays.

Query 10 is further improved by adding the intermediate data relocation. This is due to the fact that in a join operation between DBs where one of the DBs is large with more than 10 million records of data, the small data on the other DB is moved and processed together with the large data, thereby reducing the communication cost. As a result, it is shown that communication delays for queries involving large-scale data communication, which could not be avoided by data placement analysis, can be avoided by the

intermediate data relocation. This can be said that the combination of these three methods is effective in a virtual integration environment with data virtualization.

4.7.4.2 Discussion for each query

The three methods do not avoid significant delays for query 8. The three methods improve the delay of query 8 from 328x to 172x, but the delay is still high, especially degrading it by intermediate data relocation. This query includes a Left Outer Join of two tables, which has about 6 GB of input, the other about 30 GB, and the output about 5 GB. After the join operation, there is an aggregation process, and the final result of the query is only about 70KB (4,903 records). The cause of the large delay can be attributed to both the data placement analysis and the relocation of intermediate data for data virtualization. For the data placement analysis, the query 8 can be improved by improving the data placement, and that there is a possibility of improvement in the data placement analysis. On the other hand, it cannot be improved by using other intermediate data relocation methods of data virtualization. The details are described below.

With regard to data placement analysis, it is not always possible to aggregate all tables with large-scale join relations into a single DB due to storage capacity limitations. This evaluation targets a join operation that generates more than 10 million records as a large-scale join operation, and places the pairs of tables, which are frequently joined, in the same DB in priority. Therefore, the communication cost associated with the join operation is only used to select the target join operation, and the effect of infrequent join operations with particularly large communication cost is not taken into account. In the query 8, the other fact data and the master data correspond to the pairs of tables that are joined infrequently but with particularly large amounts of communication. Taking this influence into account, in a 3DB-PA environment, the other fact data on DB server 2 used in query 8 is moved to DB server 1 where the master data is located, and instead of this according to the capacity constraints, the fact data of the equipment logs on DB server 1 used in query 5 is moved to DB Server 2. We re-measured each query in this changed environment. Note that queries other than queries 5 and 8 were not affected by this change. The re-measurement results showed that query 8 significantly improved the execution time ratio to the DB-Org by a factor of 1.06. This result suggests that there is a possibility to improve data placement analysis by taking into account join operations that are infrequent and involve a large amount of communication.

The overhead for intermediate data relocation is determined based on communication volume, so other overheads such as storage processing are not taken into account. The communication volume for query 8 is 6GB x 2 + 70KB = 12GB between the data virtualization system and each DB when relocating intermediate data. It is 30GB + 6GB = 36GB when performing join processing in the data virtualization system. Although the intermediate data relocation can be expected to reduce the communication volume, the transfer process to DB in the data virtualization system, temporary storage writes to DB (COPY), and temporary storage read during execution of a subquery including the join process cannot be ignored. As a result, the performance degrades more than when processing in a data virtualization system without the intermediate data relocation. As a countermeasure, the above overhead should be included in the conditions for the cost model in the intermediate data relocation to make the decision more strictly. However, although the measurement of this cost would only prevent further performance degradation due to the intermediate data relocation, it would not improve the performance degradation fundamentally. Another possible improvement way would be to introduce 2-way Semijoin, one of the intermediate data relocation methods. In the 2-way Semijoin, the join key and the result just after the join have a significant impact on the communication volume. The respective communication volume for this query is 2 GB for the join key and 5 GB for the result after join. In other words, the communication volume is 2GB x 2 + 5GB x 2 = 14GB, which is larger than Ship Join, the intermediate data relocation method used in this implementation. Therefore, no improvement can be expected for this query.

Next, we analyze the queries other than query 8. Figure 4-12 shows the execution results for 11 queries on DB-Org and Presto with the three methods, excluding query 8. The bar graph shows the execution time for the DB-Org executed directly in Vertica and the data virtualization system with the three methods enabled on the 3DB-PA, and the line graph shows the ratio of these execution times. Seven queries (queries 1, 2, 3, 4, 6, 7, and 12) have data gathered in a single DB server even on the 3DB-PA, and all operations are executed in one of the DB servers by the query pushdown. The execution time ratio of queries 4, 6, 7, and 12 is worse than that of queries 1, 2, and 3 because of the relatively large communication volume in contrast to the short execution time. In particular, the result of query 12 is large (about 340,000 results), and the communication between the data virtualization system and the DB server causes a larger delay than the

other queries.



Figure 4-12: Execution time comparison of a single DB and data virtualization with three methods by the evaluation queries except for query 8.

Queries 9, 10, and 11 are queries that process the data across different DB servers and the intermediate data relocation works for them. However, the delays of these results are about 1.1 to 2.5 times compared to the execution on DB-Org. The reason why the delays of queries 9 and 11 are higher than that of query 10 is that their execution times are shorter and are relatively more affected by the communication of the intermediate data relocation.

Query 5 has a delay of 3.87 times that is relatively higher delay than the others. This is because all join operations are performed by the data virtualization system due to data placement. The reason why intermediate data relocation is not performed is that it is determined that the communication volume can be reduced by not performing it based on the input/output data size of the join operation. On the other hand, as shown in Table 4-2, the join operations that are processed between DB servers are for small-scale data, with a maximum size of approximately 250 MB (370,000 records), which is relatively

small, and thus do not cause as a large delay as the previously mentioned query 8. As with query 8, the delay in query 5 can be avoided by devising a data placement analysis, and the delay is improved from 3.87 times to 1.33 times by the additional evaluation with the data placement change.

Although this evaluation targets ad-hoc query execution in isolation, if batch execution of queries or concurrent execution of multiple queries is taken into account, caching of intermediate results in data virtualization systems [94] and workflow management that integrates processing common to multiple queries [95] can be considered. In addition, while this chapter used the query log analysis only for static DB dividing, dynamic query placement analysis [72] in response to changes in query trends should also be considered.

4.8 Summary

This chapter shows that the communication bottleneck in data virtualization can be resolved by three techniques: data placement analysis, query pushdown, and intermediate data relocation. Evaluation experiments shows that queries for data analysis in real data can be executed without significant latency (>4x) for 11 of 12 queries, compared to the single DB case. The remaining query can also be avoided by data placement analysis that takes into account the infrequent but heavily communicated join operation. Since this eliminates the communication bottleneck, we believe that an architecture in which a single DB is divided into multiple DBs and integrated virtually by data virtualization is effective. This would also reduce the complexity of resource design and increase maintenance costs, unlike a single DB.

Chapter 5
Evaluation of Vehicle Routing Problem using Quantum Annealing

The Vehicle Routing Problem (VRP) is a problem to optimize the delivery sequence of multiple delivery vehicles. This problem can be applied to various real-world problems such as delivery sequence in logistics and vehicle dispatch planning in demand transportation services. VRP is one of the representative combinatorial optimization problems called NP-Hard, and it is known to be difficult to find an exact solution in polynomial time using classical computers. Quantum annealing is expected to solve such combinatorial optimization problems in real time. This chapter evaluates quantum annealing on a basic VRP in order to reveal the effect of quantum annealing on a VRP with the scale of a demand transportation service. Based on the evaluation results, this chapter also discuss issues and conditions for utilizing quantum annealing in VRPs assuming real-world applications.

The structure of this chapter is as follows: Section 5.1 provides an overview of this study, Section 5.2 describes the problem setup of VRP and the formulation for implementing VRP with quantum annealing, Section 5.3 evaluates VRP with quantum annealing, Section 5.4 discusses the use of quantum annealing in VRP based on the evaluation results, Section 5.5 describes related research, and Section 5.6 concludes.

5.1 Introduction

5.1.1 Background

The Vehicle Routing Problem (VRP) is a combinatorial optimization problem for optimizing the delivery sequence of multiple delivery vehicles and is a generalization of the traveling salesman problem [97]. VRP is known as an NP-Hard problem, which is

difficult to solve in polynomial time using classical computers. In this study, as an application of VRP, we focus on a vehicle dispatch planning problem for a demand transportation service that dispatches vehicles according to passenger demand. This dispatch planning problem is a very complex problem for VRP [98][99]. It requires constraints on departure and arrival times, sequential relationships between arrival points (get on / get off), and the number of people who can be on-board. In addition, various service constraints (e.g., support for relay points, allowable delay time, etc.) arise depending on the type of service.

Recently, quantum annealing is expected to be utilized for combinatorial optimization problems that are computationally difficult to solve with classical computers. Quantum annealing is a technique for heuristically solving NP-hard combinatorial optimization problems using quantum effects [19]. With the appearance of a machine implementing quantum annealing announced by D-Wave Systems in 2011 [20], the possibility of applying quantum annealing to real business problems has emerged.

Quantum annealing can be used by formulating the combinatorial optimization problem to be solved in QUBO, as described in Section 2.3.1. Therefore, VRP can be applied to quantum annealing by formulating the objective function and constraints of VRP in QUBO formula in Eq. (2-3). However, current implementations of quantum annealing are limited in the number of qubits available due to the limited interaction relationship between qubits. For example, D-Wave Systems' machine has a sparse graph structure called a Chimera graph or Pegasus graph, which requires graph transformation when the problem involves complex interaction relationships between qubits. This significantly increases the number of qubits used to construct the problem on the quantum annealing machine [101]. Since the vehicle dispatch planning problem for demand transportation service is a VRP that requires complex constraints, and the complexity of the interaction relationship among the qubits. It may be difficult to apply quantum annealing to actual services of this problem due to the limitation of the number of qubits.

5.1.2 Contributions

This chapter evaluates the performance of quantum annealing for a VRP with limited constraints in order to verify the application of quantum annealing to demand transportation services. The results show that the optimal solution is obtained in about

40 ms for a VRP with two vehicles and four locations. However, since the current hardware has a limit to the number of qubits, the performance of the VRP when scaled up to the scale of a demand transportation service cannot be evaluated. To evaluate the scaling of quantum annealing, we used two types of Ising machine that simulates the quantum annealing on a classical device. In this evaluation, by using data of three different scales assuming a demand transportation service, the performance of the VRP on the Ising machines is compared with that on an exact solution method of brute-force search and a heuristic approximate solution solver, which are conventional classical methods. This comparative evaluation confirms that the Ising machine improves accuracy by up to 10% compared to the approximate solution solver in terms of the objective function at scales where the exact solution method cannot solve. On the other hand, the Ising machine is inferior to the approximate solution solver in terms of execution time by more than two orders of magnitude. Based on these evaluation results, we discuss the issues and summarizes the requirements for utilizing quantum annealing in the VRP for real-world applications.

The contributions of this research are the construction of the QUBO formula for basic VRP, the evaluation of quantum annealing with data assuming real service use, and the organization of the issues and machine utilization requirements for quantum annealing utilization in VRP assuming real use, based on the evaluation results.

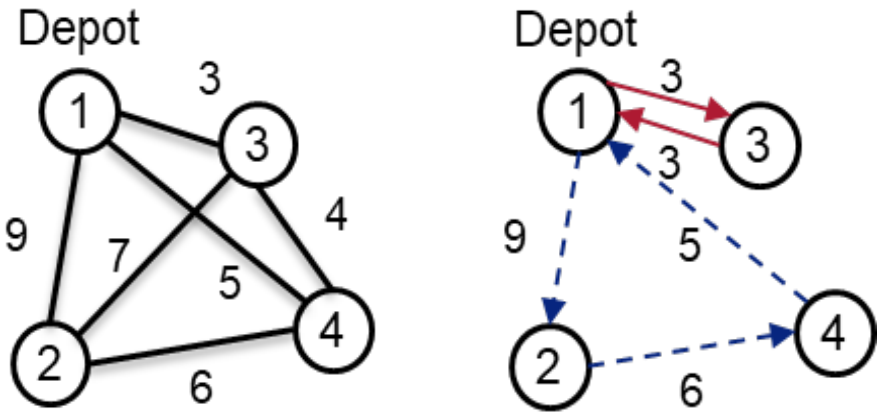
5.2 Applying Quantum Annealing to VRP

5.2.1 Problem Definition

The problem in this study is a combinatorial optimization problem to find a route that minimizes the total travel cost when multiple vehicles visit all the target locations as a basic VRP. In addition, the problem assumes that all vehicles start from a base location called Depot and always return to Depot at the end. The travel cost is based on the distance between the locations. The travel costs, which are necessary for solving VRP, are assumed to have been calculated in advance.

As an example problem, let us consider a route for two vehicles and four locations. As shown in Figure 5-1 (a), the travel costs between all locations are calculated in advance. Here, Depot 1 is the departure location and arrival location of the vehicle. Figure 5-1 (b) can be calculated as one of the optimal solutions. The delivery vehicle in the red line

takes route 1->3->1 with a travel cost of 6, and the vehicle in the blue dotted line takes route 1->2->4->1 with a travel cost of 20, for a total travel cost of 26.



(a) Travel costs between all locations

(b) An example of optimized routes for two vehicle

Figure 5-1: VRP with two vehicles and four locations

5.2.2 Mathematic Model

In order to formula the QUBO of Eq. (2-3), which takes a binary variable to apply quantum annealing, the decision variable of VRP is defined as follows.

$$x_{ps}^{(v)} = \begin{cases} 0 & \text{A vehicle } v \text{ is not staying } s^{\text{th}} \text{ at a location } p, \\ 1 & \text{A vehicle } v \text{ is staying } s^{\text{th}} \text{ at a location } p. \end{cases} \tag{5-1}$$

Let V be the number of vehicles, P be the number of locations, and S be the maximum order of visits, then vehicle v, location p, and visit order s have the following values:

$$1 \le v \le V, \qquad 1 \le p \le P, \qquad 1 \le s \le S \tag{5-2}$$

Let d_{ij} be the travel cost of moving from a location i to a location j, then the objective function of the VRP is expressed below as a minimization problem for a set X of decision variables $x_{ps}^{(v)}$ defined in Eq. (5-1).

$$\underset{X}{\text{argmin}} \sum_{v=1}^{V} \sum_{i=1}^{P} \sum_{j=1}^{P} \sum_{s=1}^{S-1} d_{ij} x_{is}^{(v)} x_{j(s+1)}^{(v)} \tag{5-3}$$

When a location where a vehicle v is staying changes by shifting a visit order from s

to $s+1$, it can be judged as a move of the vehicle v. Eq. (5-3) adds a travel cost d_{ij} between location i and j when a vehicle stays at location i in visit order s and then stays at location j in the next visit order, $s+1$, as a move. Note that $d_{ij}=0$ for $i=j$, since the location has not been changed. Figure 5-2 depicts the values of the set of decision variables X in the example of the optimal route with two vehicles and four locations shown in Figure 5-1 (b). For example, a move is detected when $v=1, s=1, i=1, j=2$, and $d_{12}=3$ is added as the travel cost using the formula (2-3).

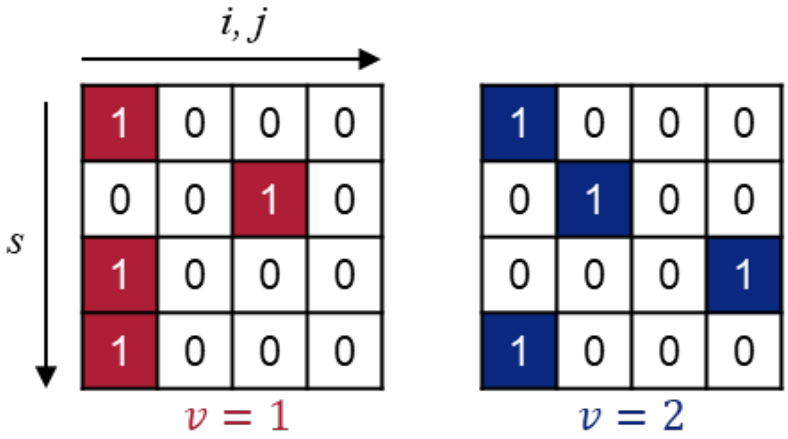


Figure 5-2: Binary values in a set of decision variables X in Figure 5-1 (b).

As a constraint condition of the VRP, assuming that location 1 is the Depot, the visit order 1 and S in each vehicle must always stay at the location 1. Therefore, the following constraints arise.

$$\forall v, \quad x_{11}^{(v)} = 1, \tag{5-4}$$

$$\forall v, \quad x_{1S}^{(v)} = 1. \tag{5-5}$$

The VRP requires that all locations are visited by one of the vehicles. Therefore, it is necessary to constrain that all locations are always visited once by one of the vehicles and in one of the visit orders. However, since Depot is visited at least twice according to the constraints in Eq. (5-4) and Eq. (5-5), Depot is excluded from the constraints. The following constraints are added based on the above.

$$2 \le p \le P, \quad \sum_{v=1}^{V}\sum_{s=1}^{S} x_{ps}^{(v)} = 1. \tag{5-6}$$

In addition, the decision variable $x_{ps}^{(v)}$ in Eq. (5-1) has the possibilities that a vehicle

disappears at a certain visit order or exists at more than one location at the same time. To eliminate these possibilities and to ensure that all vehicles always visit a single location in each visit order, the following constraints are added.

$$\forall v, \forall s, \qquad \sum_{p=1}^{P} x_{ps}^{(v)} = 1. \tag{5-7}$$

After satisfying all the above four constraint terms, Eq. (5-4), (5-5), (5-6), and (5-7), the route of each vehicle that results in the least cost is calculated from Eq. (5-3), which is the objective function.

5.2.3 QUBO Model

To perform the VRP with quantum annealing, the QUBO equation is constructed. The Hamiltonian $\mathcal{H}$ in Eq. (2-3) is defined as follows from the objective function and constraints defined in section 5.2.2.

$$\mathcal{H} = \mathcal{H}_A + \lambda_1 \mathcal{H}_B + \lambda_2 \mathcal{H}_C + \lambda_3 \mathcal{H}_D + \lambda_4 \mathcal{H}_E, \tag{5-8}$$

$$\mathcal{H}_A = \sum_{v=1}^{V} \sum_{i=1}^{P} \sum_{j=1}^{P} \sum_{s=1}^{S-1} d_{ij} x_{is}^{(v)} x_{j(s+1)}^{(v)}, \tag{5-9}$$

$$\mathcal{H}_B = \sum_{v=1}^{V} \left(1 - x_{11}^{(v)}\right)^2, \tag{5-10}$$

$$\mathcal{H}_C = \sum_{v=1}^{V} \left(1 - x_{1S}^{(v)}\right)^2, \tag{5-11}$$

$$\mathcal{H}_D = \sum_{p=2}^{P} \left(1 - \sum_{v=1}^{V} \sum_{s=1}^{S} x_{ps}^{(v)}\right)^2, \tag{5-12}$$

$$\mathcal{H}_E = \sum_{v=1}^{V} \sum_{s=1}^{S} \left(1 - \sum_{p=1}^{P} x_{ps}^{(v)}\right)^2. \tag{5-13}$$

$\mathcal{H}_A$ corresponds to the objective function, Eq. (5-3), where the travel cost is added as the vehicles move. Thus, quantum annealing changes the values of the decision variables corresponding to the qubits so as to minimize the total travel cost. $\mathcal{H}_B, \mathcal{H}_C, \mathcal{H}_D$ and $\mathcal{H}_E$ correspond to the constraint terms, Eq (5-4), (5-5), (5-6), and (5-7), respectively. Each of these equations increases $\mathcal{H}$ by a positive integer due to the square of the parentheses

when the constraint is violated, so that quantum annealing changes the value of the decision variables to avoid the violation.

A constraint violation in Eq. (5-8) is an incorrect solution because violation of any constraint term contradicts the problem setup as a VRP. By making the penalty coefficients $\lambda_1, \lambda_2, \lambda_3$ and λ_4 in Eq. (5-8) sufficiently larger than the values that $\mathcal{H}_A$ can take, the probability of constraint violation can be reduced. On the other hand, extremely large penalty coefficients make the convergence of $\mathcal{H}_A$ unstable, and it takes time to obtain a solution with low total travel cost. Therefore, the penalty coefficient must be tuned to be small enough to prevent constraint violations. Note that it is possible to calculate a standard value for the penalty coefficient depending on the problem to be solved [102]. In this VRP, the initial value of each penalty coefficient is set to $\max(d_{ij})$, and tuning is performed to increase the penalty coefficient on the condition that a feasible solution ratio, which means the probability of obtaining a feasible solution (a solution with no constraint violation), reaches a target value or that the probability stops increasing. In this case, the number of violations is measured for each constraint term, and the penalty coefficient for the constraint term with the most violations is increased first. If the feasible solution ratio satisfies its target value, its penalty coefficient for the constraint of the smallest number of violations is decreased.

5.3 Evaluation

This section evaluates quantum annealing applying to VRP for the problem setup described in Section 5.2.1 as a basic VRP with limited constraints. The QUBO formula in Eq. (5-8) is used to evaluate the performance of the quantum annealing to solve the VRP by running it on a quantum annealing machine. However, the current implementation of the quantum annealing has a significant limitation in the scale of applicability, so in order to evaluate its applicability to real services, we used Ising machines that can run on a relatively large scale and can process the Ising model as well as quantum annealing. This evaluation measured the objective function, the total travel cost value, and the execution time to derive the solution. By comparing the results with existing classical methods for the VRP, the applicability of quantum annealing is discussed in terms of a scaling perspective. In addition, the results are evaluated using multiple data scales that are assumed to be the scale of actual demand transportation

services.

5.3.1 Evaluation Environment

D-Wave 2000Q [103], a machine that implements quantum annealing in hardware, is used to evaluate the performance of quantum annealing. To use this machine in the cloud, we used the API of the D-Wave Ocean SDK [104]. Its version used in this evaluation is 1.6.0. Since the qubits implemented in the D-Wave 2000Q are composed of a sparse graph structure called Chimera graph, the QUBO used in this evaluation is transformed by the graph embedding method called "minor embedding" [100] provided by the D-Wave Ocean SDK. The D-Wave 2000Q has 2048 qubits available, but the graph embedding reduces the number of available decision variables to 64 when the interaction between all decision variables in the problem formulated in QUBO is full coupling. In this evaluation, 'DW_2000Q_VFYC_5' is used as the solver of the D-Wave 2000Q, with an 'optimization' option to allow automatic tuning of the machine parameters such as *chain_strength*. The D-Wave Ocean SDK version 1.6.0 is used.

For the scaling evaluation, we use Fujitsu Digital Annealer (DA) [105] and OpenJij [106] as the Ising machines that can process the Ising model as well as quantum annealing. The DA is a machine dedicated to solving combinatorial optimization problem, which implements the Ising model processing algorithm in digital circuit [107]. This can be used via a Web API provided by Fujitsu. In this evaluation, the Web API of 'FujitsuDAPTSolver' is used as the solver, which automatically tunes some machine parameters. The number of available bits used for decision variables is 1024 bits. OpenJij is an open-source software providing simulated quantum annealing (SQA) for Python, which simulates quantum annealing. Its version used in this evaluation is 0.0.8. Both DA and OpenJij can use the full coupling of interactions between all variables, allowing QUBO to be executed as is.

In order to compare these Ising machines with existing VRP methods, the scaling evaluation uses an exact solution method based on a brute-force search and a heuristic approximate solution method. The exact solution method is implemented in Python by scratch, and uses depth-first search to determine the combination of locations that each vehicle visits, and then identifies the visit order in which each vehicle visits from the combination of locations by permutation. The VRP solver from Google OR-Tools [108] is used for the approximate solution method in Python. The VRP solver utilizes a local

search method that combines dynamic programming and tree search, and always outputs the same solution under the same execution conditions.

For the evaluation using OpenJij's SQA, exact solution method, and approximate solution method, the programs are executed on a general-purpose server. The used server is a Dell PowerEdge R410 with the following hardware specifications: CPU: Xeon X5675 (3.06GHz, 6 Cores) x 2, Memory: 96GB, SAS-HDD: 1TB x 4 (RAID 1), NIC: 1000Base-T, and with software specifications: OS: CentOS 6.6, Python: 3.6.0.

5.3.2 Evaluation Data

This evaluation uses data with three different scales of the problem, based on the number of delivery vehicles (V) and the number of locations (P), assuming an actual demand transportation service. The maximum value of the visiting order S is $P + 1$, since the maximum pattern is a situation where a single vehicle visits all locations and Depot visited twice in both the start and the end. However, in consideration of practical issues, it is necessary to maintain fairness by using multiple vehicles to visit. In addition, the Ising machines including quantum annealing machine have an upper limit on the number of bits, and reducing S to reduce the number of decision variables will increase the size of the problem that can be performed on the Ising machines. Therefore, when the average number of locations per vehicle is 3 or more, the maximum value of S for the order of visits is set as follows using the number of delivery vehicles V and the number of locations P.

$$S = \begin{cases} \left\lfloor \frac{3P}{V+2} + 0.5 \right\rfloor, & if\ \frac{P}{V} \geq 3.0 \\ P, & otherwise \end{cases} \tag{5-14}$$

As data for evaluating the size of the three different problems, 12 different sets of data are prepared for the values of V, P, and S shown in Table 5-1. The number of bits of the decision variable required by the Ising machine is also listed. Note that $V = 2$ and $P = 4$ are used only for evaluating the performance of quantum annealing with D-Wave 2000Q.

Table 5-1: Scale of VRP problem in evaluations (V: # of vehicles, P: # of locations, S: max values of visit order).

	V	P	S	# of variables
*	2	4	4	32
Small-scale	2	10	8	140
		11	9	176
		12	10	216
		13	10	260
Middle-scale	4	19	10	760
		20	10	800
		21	10	840
		22	11	968
Large-scale	6	40	15	3,600
	10	160	40	64,000
	15	400	71	426,000

* Evaluation data for quantum annealing

The distances between locations are required as data to solve the VRP. In this evaluation, the distances between locations are generated according to the number of locations at each of the scales mentioned above. First, the coordinates [X, Y] of each location are randomly generated. The range of X and Y coordinates is an integer between 0 and 1000. A coordinate of the Depot is set to [500,500]. For these randomly generated coordinates, we calculated the Euclidean distance between all two coordinates and used it as the distance between the locations. Note that the D-Wave 2000Q used in the evaluation of quantum annealing is limited to a resolution of 5 bits for the scale of the interaction between qubits, so the distance between the locations is recalculated by a factor of 1/50 only for the data used in the evaluation of quantum annealing.

Based on these evaluation data, the penalty coefficients described in Section 5.2.3 were tuned in advance to the values shown in Table 5-2 for each annealing machine. The target feasible solution ratio, which is a probability of obtaining a feasible solution, was set to 80%, and tuning was performed with V=2, P=10 for the small-scale problem and V=4, P=19 for the middle-scale problem. Other problems in each scale uses these penalty coefficients as a common parameter. Note that the D-Wave 2000Q did not reach the target feasible solution ratio and the increase of the feasible solution ratio stopped; DA and OpenJij reached the target value. For OpenJij, the same penalty coefficients is used

for small-scale and middle-scale problems.

Table 5-2: Penalty parameters of each annealing machine.

	λ_1	λ_2	λ_3	λ_4
D-Wave 2000Q	55	42	42	41
DA-small-scale	750	750	875	750
DA-mid-scale	1000	1000	1250	1000
OpenJij	1500	1300	1230	1250

5.3.3 Performance Evaluation of Quantum Annealing

We evaluate the performance of VRP in quantum annealing using D-Wave 2000Q. Since graph embedding for the interaction between qubits implemented in D-Wave 2000Q allows the use of up to 64 qubits, the evaluation data of two vehicles and four locations is used in this evaluation. The time for the annealing process to actually search for a solution can be specified as a parameter during execution, called annealing time, and the parameters in this evaluation are the minimum value of 1 μs and 10 μs. D-Wave 2000Q can perform multiple annealing processes in a single execution. In this evaluation, the number of annealing processes is set to 100 times. This means that 100 solutions can be obtained in one execution of D-Wave 2000Q.

The results of the executions on the D-Wave 2000Q in the environment described above confirmed that the exact solution could be obtained for both the 1 μs and 10 μs annealing times. The feasible solution ratio for each annealing time was 41% for 1 μs and 52% for 10 μs. Thus, the stability to obtain the correct solution increases by increasing the annealing time.

Figure 5-3 shows the execution time and its breakdown in the two annealing times. The definitions of the breakdown for each execution time are as follows.

- *qpu_programming_time*: Time to set the external field coefficients of each qubit and the interaction between qubits in QUBO to the device.
- *qpu_anneal_time*: Total time for annealing process.
- *qpu_readout_time*: Total time to read the result of annealing process.
- *qpu_delay_time*: Total time to reconfigure the device environment for the

annealing process.

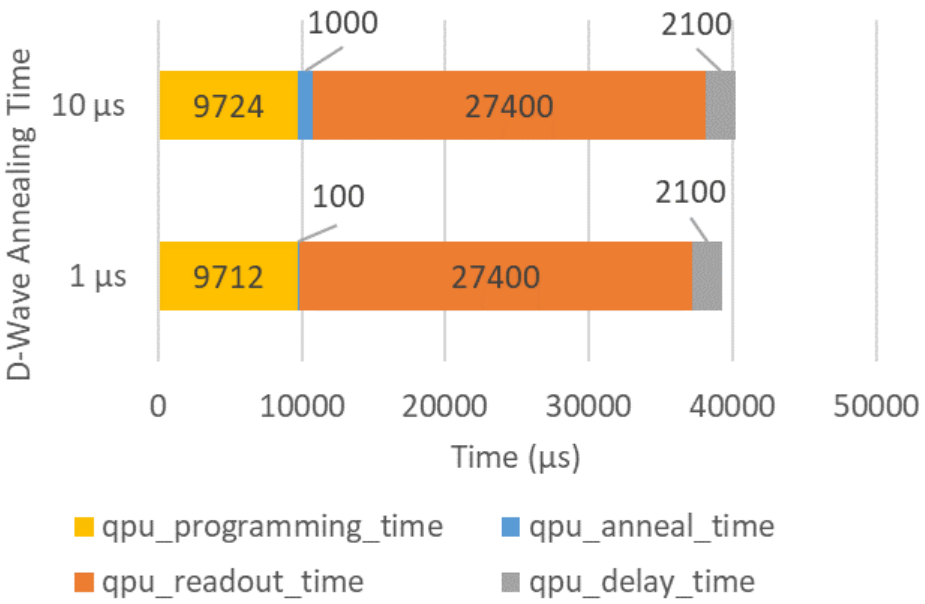


Figure 5-3: Execution time on D-Wave 2000Q

The time measured in *qpu_programming_time* is performed only once in a D-Wave 2000Q run, while *qpu_anneal_time*, *qpu_anneal_time*, and *qpu_delay_time* represent the sum of 100 times the time measured for each annealing process. The *qpu_anneal_time*, which is the time for 100 annealing processes specified at runtime, is very short compared to the other processing times. This can be said that the execution time of D-Wave 2000Q is dominated by non-annealing operations. It can be seen that the current machine has a relatively large non-annealing processing overhead compared to the annealing processing time. Comparing the execution times of 1 µs and 10 µs, there is a 10-fold difference in the *qpu_anneal_time*, as shown in the ratio of annealing times, but the processing times other than the *qpu_anneal_time* are almost fixed values independent of these annealing times. Therefore, even if the annealing time is increased to improve the stability to obtain the correct solution, the overall machine execution time is not significantly delayed, which is an advantage from a scaling perspective.

5.3.4 Scaling Evaluation

D-Wave 2000Q, a quantum annealing machine, cannot be evaluated on data of the scale of a demand transportation service due to the limitation of available qubits. On the other hand, an Ising machine implemented on classical devices can run on a larger scale of data. In this section, we evaluate the practicality of quantum annealing for VRP on a larger scale by comparing values of the objective function and execution time with conventional methods for VRP by using Ising machines DA and OpenJij.

In this scaling evaluation, the Ising machines was run 100 times as well as the evaluation of quantum annealing. Both DA and OpenJij can obtain about 80 feasible solutions by tuning the penalty coefficients. A solution with the smallest value of the objective function among the feasible solutions obtained is evaluated, assuming that it is used for the VRP application as the solution. The Ising machine allows the user to specify the time of one annealing process as the number of steps. Increasing the number of steps increases the probability of obtaining a good solution, but the execution time increases in proportion to the number of steps. In this evaluation, the number of steps used in tuning the penalty coefficient is used as the initial value, and 100 times of annealing process is executed multiple times by increasing the number of steps for each data. The result of the minimum number of steps at the point when the minimum value of the objective function no longer changes is adopted. The minimum value of the objective function and the execution time of 100 times of annealing are evaluated. Note that the conventional method of the VRP uses the results of only one run, since the solution obtained is fixed.

5.3.4.1 Objective Function

The total travel cost of the objective function in VRP is evaluated in this section. Figure 5-4 shows the results of executing a small-scale problem with the exact solution method, approximate solution method, DA, and OpenJij. The approximate solution method and a part of OpenJij have larger total travel costs than the exact solution method. On the other hand, the results for DA are the same as for the exact solution method. Figure 5-5 shows the results of executing a middle-scale problem. The total travel cost for DA is about 10% lower than that for the exact solution method, and for OpenJij, it is about the same or about 5% lower than that for the approximate solution method. Large-scale problems are omitted because DA and OpenJij were unable to perform them. The

reason for this is that DA could not be run due to a lack of bits, and OpenJij was dropped due to execution time.

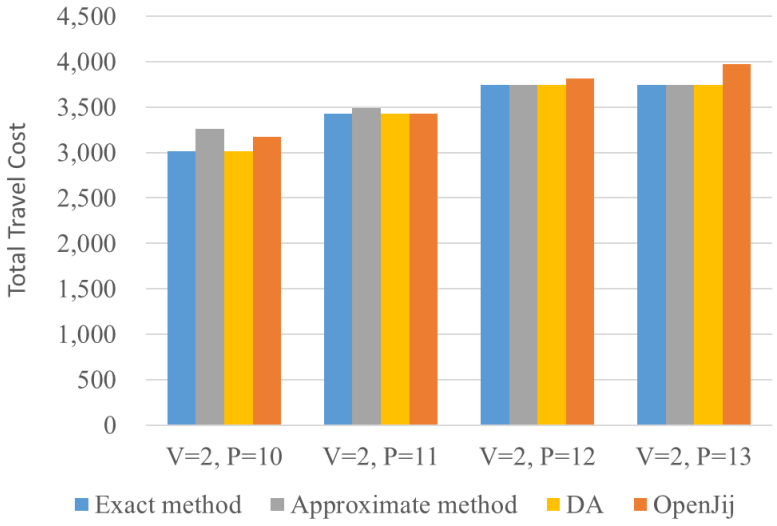


Figure 5-4: Total travel cost in small-scale problems.

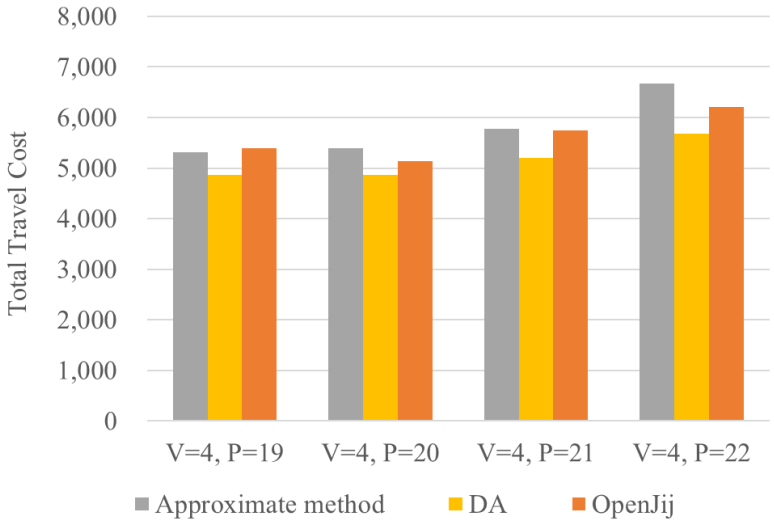


Figure 5-5: Total travel cost in middle-scale problems

DA results in stable convergence to low values of the objective function. OpenJij is able to achieve lower values of the objective function compared to the approximate solution, but its convergence is less stable than the other results. The advantage of both

DA and OpenJij over the approximate solution increases with the size of the problem. Although these are simulators of the quantum annealing, the behavior of convergence to the best solution according to the Ising model is similar to that of quantum annealing, and it can be said that quantum annealing may be superior to the approximate solution method in terms of values of the objective function. On the other hand, there is a limit to the scale of problems that can be solved with Ising machines on classical devices.

5.3.4.2 Execution Time

The execution time in VRP is evaluated in this section. The execution time used here is the time to calculate the total travel cost in previous section. Note that the measured time is the processing time of the Ising model on each machine and does not include the communication time in the DA's Web API or the waiting time in the queue. Figure 5-6 shows the measured execution times for all three problem sizes. The exact solution method was able to execute only small-scale problems, while DA and OpenJij were able to execute both small- and middle-scale problems. Only the approximate solution method was able to execute large problems.

The exact solution method is faster than OpenJij for the smallest problem in small-scale problems, but the gradient with scale is large. And as the number of locations increased, the execution time of the exact solution method exceeds that of OpenJij, requiring several hours of execution time. The approximate solution method, DA, and OpenJij has similar gradients for an increase in the number of locations within the same scale, but OpenJij's execution time increased significantly from small to medium scale. The gradient of DA is similar to that of the approximate solution method, but the absolute values differ by two orders of magnitude.

Compared to the results of the exact solution methods, DA and OpenJij are significantly superior as the scale increases. However, compared to the approximate solution methods, DA and OpenJij are significantly inferior, especially OpenJij. OpenJij is software, and in its current implementation, execution time increases exponentially as the number of bits increases due to its sequential execution. DA is faster than OpenJij due to the parallel execution effect of the dedicated digital circuit implementation, but it is still slower than the approximate solution methods.

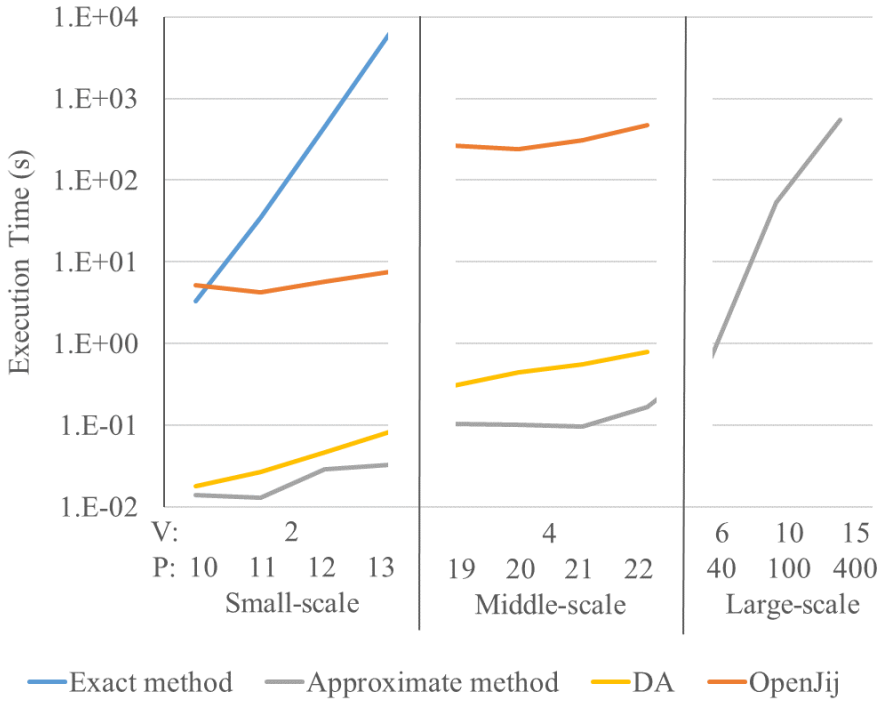


Figure 5-6: Execution time on Ising machines and conventional classical methods.

5.4 Discussion for the Use of Quantum Annealing

The result of the performance evaluation of quantum annealing described in Section 5.3.3 shows that current quantum annealing machine has strong constraints on the number of qubits and runtime overhead. However, the annealing process itself is fast, in the order of microseconds. And it is expected to be able to solve VRP more efficiently by increasing the number of applicable qubits with advances in the machine implementation. In order to solve larger problems on quantum annealing, a formulation method for reducing the number of bits in the VRP and a graph embedding method for chimera graphs have to be devised. With regard to the former formulation method, for example, the constraints for the Depot, Eq. (5-4) and (5-5), can be set as constants to reduce the number of variables. As a side effect, these constraint Eq. (5-10) and (5-11)

can be removed from the QUBO Eq. (5-8), which is expected to simplify the tuning of the penalty coefficients. For the latter, the graph embedding in this evaluation is a simple conversion of chimera graphs to all-join graphs as a general-purpose method. By devising the graph structure of the VRP decision variables as the problem setup, it is possible to increase the scale of the available problems. In other words, it eliminates the simple conversion to all-joining and improves the efficiency of graph embedding. For example, eliminating edges between physically distant locations can be used to create subgraphs. Since the number of used physical qubits does not change in this method, the processing time remains the same. However, since the scale of the problem is larger, it is necessary to increase the time and number of annealing processes to ensure the accuracy of the values of the objective function. In contrast, it is found from this evaluation that the overhead other than the time of the annealing process does not change. Therefore, it is considered that the execution can be performed with a delay of microseconds compared to the overall execution time of 40 ms in this evaluation. This means that the effect of the devised graph embedding is expected.

In terms of scaling of quantum annealing, the results of the evaluation of the objective function using the Ising machines in Section 5.3.4.1 indicate that quantum annealing may be superior in real applications of VRP, which requires accuracy, even when the number of physical qubits of quantum annealing is increased. On the other hand, in terms of execution time, for a middle-scale problem evaluated with the Ising machines, the execution time is on the order of 10 seconds for DA and 10,000 seconds for OpenJij, which is very slow compared to the 0.1 second order of the conventional classical method that find approximate solutions. At current stage, it is not possible to quantitatively evaluate the execution time of quantum annealing on this scale due to implementation limitations. However, since DA and OpenJij showed advantages in terms of accuracy of the objective function, it is expected that the practicality of quantum annealing in VRP will increase as the evolution of quantum annealing machines brings this difference in execution time closer. In particular, the approximate solution method shows a significant delay in large-scale problems, so it is possible that quantum annealing, including Ising machine with classical devices, will show an advantage when applied to large-scale problems or when the number of qubits used in VRP is reduced, as mentioned above.

Based on the results of the evaluation up to this point, we summarize the conditions for utilizing quantum annealing in VRP. If quantum annealing is available on a feasible

scale, it should be utilized in terms of the accuracy of the objective function and the execution time. In order to expand the range of the feasible scale, further work should be done on the formulation of QUBO and graph structure. For larger scales, the machine to be used will depend on the required performance and accuracy. For real-time solutions to optimization requests, such as web services, conventional classical solvers are appropriate at current stage. If the accuracy of the objective function is required and the optimization process can be performed in batch, a relatively fast Ising machines such as DA is appropriate. Software-implemented Ising machines such as OpenJij can be run on general-purpose servers and are relatively easy to use, so they may be of use when you want to use quantum annealing on a trial basis, for example, when adding various constraints to the VRP used in this evaluation.

5.5 Related Work

Existing methods for VRP include local search, simulated annealing, and genetic algorithms as approximate solution methods, and mathematical programming and branch-and-bound methods as exact solution methods. Specifically, research has been conducted on a problem called Capacitated VRP (CVRP), which adds capacity constraints to VRP [109][110], and on a problem called VRP with Time Window (VRPTW), which adds time constraints [111][112]. These are specialized methods for their respective problems. Therefore, further innovations are needed in applying constraints to these methods for real-world applications, such as demand transportation services. From this perspective, the Ising model used in quantum annealing can be applied to combinatorial optimization problems in a generalized manner, and it is a great advantage that constraints can be added relatively easily by adding constraint terms to the QUBO formula.

Research on solving NP-hard combinatorial optimization problems using quantum annealing has been increasing in recent years [21][113][114]. In particular, studies applying quantum annealing to VRP include a study of applying CVRP as an optimization problem of delivering customers by vehicles such as buses [115], and a study of applying CVRP to the problem of adding time and vehicle state constraints to CVRP using logistics as a use case [116]. These studies have not evaluated VRP under basic conditions or with scaled data assuming demand transportation services.

5.6 Summary

In this chapter, we constructed a QUBO of a basic VRP with limited constraints, and evaluated its performance on D-Wave 2000Q, a quantum annealing machine, and the scaling of quantum annealing on two Ising machines: DA and OpenJij. The performance evaluation of quantum annealing confirmed that the optimal solution can be derived in 40 ms. Although the annealing process itself was able to derive a solution in microseconds, the scale of execution on current quantum annealing machine is small, and the overhead of processing other than annealing is relatively large. In the scaling evaluation, DA and OpenJij were compared with the exact solution method and the approximate solution method as conventional VRP methods on a scale assuming a demand transportation service, and it was shown that the approximate solution method improves accuracy by up to 10% compared to the exact solution method in terms of the objective function. On the other hand, in terms of execution time, OpenJij was inferior to the approximate solution method by more than two orders of magnitude. Based on these evaluation results, we summarized the issues and conditions for utilizing quantum annealing in VRP for real-world applications.

Even at the present stage, an annealing simulator has the potential to be superior for middle-scale problems in the application to demand transportation services, and this potential is expected to increase with the evolution of hardware implementation technology. In order to increase this possibility, formulation methods to reduce the number of qubits and graph structure transformation methods should be considered from the utilization side. In addition, when constraints are added based on the application to demand transportation services, methods that specialize in VRP with limited constraints, such as Google OR-Tools that is an approximate solution method, cannot be used as is. In this case, quantum annealing, which can be used in general-purpose applications, can be considered to be superior. Toward this realization, it is necessary to apply QUBO to CVRP and VRPTW, which are general problems with additional constraints as a realistic problem setting, and to construct and evaluate QUBO equations for problems combining other service realization conditions.

Chapter 6
Evaluating Quantum Annealing for Grouping Optimization with all Members' Compatibility

Grouping optimization problem is one of the combinatorial optimization problems that maximize group effectiveness by composing groups based on member's attributes, personality, or group statistics. This problem is appeared, for examples, in teaming for an organization or collaborative learning. In a previous work, a QUBO model for quantum annealing is proposed for a grouping optimization problem where the group effectiveness is defined by compatibility of pair members' personality or between a group and a member. However, the QUBO model cannot be applied in case that the group effectiveness is calculated by compatibility of three or more members composing groups. In this chapter, we formulate a general model of the grouping optimization for quantum annealing, and evaluate it assuming collaborative learning on quantum annealing machine in comparison with simulated annealing and hill climbing. As a result, we confirmed that quantum annealing solves this problem about 400 times faster than the hill climbing and about 1,300 times faster than simulated annealing, although the stability of solution’s accuracy on quantum annealing machine is lower than these methods.

The structure of this chapter is as follows: Section 6.1 provides an overview of this study, Section 6.2 describes the problem setup of the grouping optimization and the formulation for implementing it with quantum annealing, Section 6.3 evaluates it with quantum annealing and compares two solvers, and Section 6.4 concludes.

6.1 Introduction

6.1.1 Background

Grouping optimization is a combinatorial optimization problem that maximizes group effectiveness by composing groups. There are a variety of applications involving this grouping optimization problem. In organizational teamwork, for example, good team composition improves organizational performance. In collaborative learning, where learners work together to enhance learning in groups [117], good group composition can increase learner's satisfaction and learning efficiency. The matching problem for between two persons or between a person and an organization is also a kind of the grouping optimization problems to compose good pairs. Indicators that determine the value of composed groups are simply individual attributes of members, such as the gender, age, and skills, as well as group statistics, such as its goal and size. In addition, the compatibility of the members who compose the group is also used for the indicator, which is determined not only by the individual attributes, but also by their personalities as quantified by the Big Five [118]. The personality compatibility among group members is an important indicator in group optimization since it significantly affects group members' satisfaction [119][120].

The computational complexity of the grouping optimization explodes depending on the number of members and groups. Therefore, we focus on the use of quantum annealing to solve this problem. Figure 6-1 depicts an example of the grouping optimization in case of collaborative learning [121]. This case is the problem of dividing the learners of a learning course into groups consisting of a specific size and minimizing the course drop-out probability for each learner. The drop-out probabilities of learners are calculated by personality compatibility among group members for all candidate combinations. After this calculation, quantum annealing optimizes to select the groups from all candidate groups with the drop-out probabilities of each member. The previous work [122] formulates a grouping optimization problem to a QUBO model to solve it on quantum annealing. However, because its QUBO model focuses on a compatibility of a pair of members or between a member and a group, it cannot optimize groups with three or more members' compatibility as the case of Figure 6-1. In addition, it is also not applicable to the problem where the indicators are calculated in a black-box way such as machine learning.

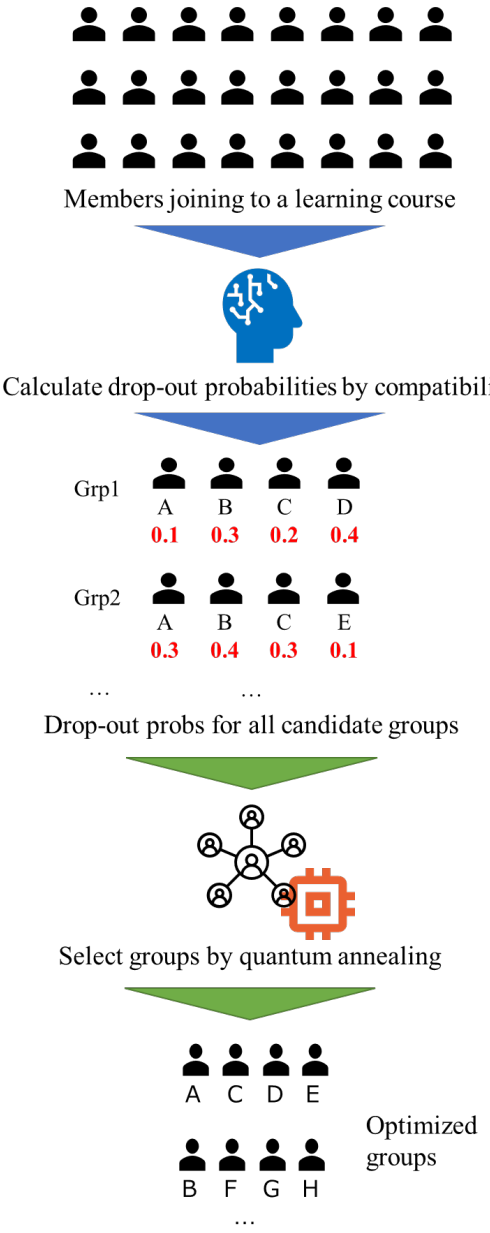


Figure 6-1: An example of grouping optimization in collaborative learning.

6.1.2 Contributions

This chapter formulates a general model of grouping optimization problem for quantum annealing in order to compose groups optimized with all members'

compatibility. Because our proposal model becomes a higher order binary optimization (HOBO) model, we transform its model to QUBO model with fewer slack variables in order to execute it on quantum annealing machine. Then, we evaluate the model on quantum annealing machine, D-Wave Advantage, in comparison with simulated annealing and hill climbing in situation of collaborative learning to compose groups using all members' personality compatibility.

The contributions of this research are the construction of the QUBO formula for a general grouping optimization problem, and its evaluation assuming collaborative learning on quantum annealing comparing with two classical methods.

6.2 Grouping Optimization using Quantum Annealing

6.2.1 Problem Settings

In the grouping optimization problem this chapter focuses on, the number of members to be optimized, the number of groups, and the number of members in a group, called group size, are determined in advance. Since all members have to be assigned to one of the groups through the optimization, the total group size has to equal the number of members. An indicator of the value of composed groups (e.g., satisfaction rate or drop-out probability, that calculated by members' compatibility) is pre-calculated for each member in all candidate groups. The candidate groups are created by the group size combinations for all members. For example, in case of creating candidate groups of 6 members for a group composing 3 members, 20 candidate groups are created as all combinations that choose 3 members for its group from 6 members. Although there is no need to calculate the indicators of candidate groups for other groups with same group size, ones for the different group size are necessary.

An objective of this grouping optimization is to minimize or maximize the indicators of all members. Whether the objective is maximization or minimization depends on the indicator. If the indicator is drop-out probability that should be minimized as Figure 6-1, the objective would be minimization problem.

6.2.2 Mathematical Model

We formulate a mathematic model of the grouping optimization problem mentioned above. Symbols for the formulation in this paper are listed in Table 6-1. The objective

function of this problem is formulated over the set X of binary decision variable $x_{g,m}$:

$$\underset{X}{\text{argmin}}\ or\ \underset{X}{\text{argmax}} \sum_{g \in G} \sum_{c \in C_g} \left(\left(\sum_{m \in M_c} D_{c,m} \right) * \prod_{m \in M_c} x_{g,m} \right). \tag{6-1}$$

In this formula, the selection of a particular group $g \in G$ from the candidate groups C_g can be indicated only if all the members M_c belonging to its candidate group $c \in C_g$ are selected. In this case, all the indicators $\sum_{m \in M_c} D_{c,m}$ for the selected candidate group are added.

Table 6-1: Symbols to formulate the grouping optimization problem.

Symbols	Descriptions
$\boldsymbol{G}$	Set of Groups.
$\boldsymbol{N_g}$	Number of members in a group $g \in G$.
$\boldsymbol{M}$	Set of members.
$\boldsymbol{C_g}$	Set of candidate groups as all combinations that choose members for a group $g \in G$ from all members in M.
$\boldsymbol{M_c}$	Set of members assigned to a combination c $\in C$.
$\boldsymbol{D_{c,m}}$	Indicator of a member $m \in M_c$ in a candidate group c $\in C$.
$\boldsymbol{x_{g,m}}$	Binary decision variable if a member $m \in M_c$ is assigned a group $g \in G$.

There are two constraints for this problem to correctly work the objective function in Eq. (6-1). The first constraint is that a member $m \in M$ must assigned to a single group; neither not assigned nor assigned two or more groups is not allowed. This one-hot constraint for each member is formulated as:

$$\forall m \in M, \quad \sum_{g \in G} x_{g,m} = 1. \tag{6-2}$$

The second constraint is that the number of members assigned to a group $g \in G$ must be equal to group size in that group, N_g. This group size constraint for each group is formulated as:

$$\forall g \in G, \quad \sum_{m \in M} x_{g,m} = N_g. \tag{6-3}$$

Our model for the grouping optimization consists of the objective function in Eq. (6-1) and the two constraints in Eq. (6-2) and Eq. (6-3). This model is to solve a problem

by selecting a pre-determined number of groups from a set of candidate groups, although the binary decision variable is defined by a matrix of members and groups in order to reduce the number of this variables. The number of candidate groups $|C_g|$ for a group $g \in G$ is the number of combinations that chooses members in group size N_g from all members M, which is calculated as follows:

$$|C_g| = \binom{|M|}{N_g} = \frac{|M|!}{N_g!\,(|M| - N_g)!}, \tag{6-4}$$

where the $|M|$ is the number of members M. The number of all combination of this problem O is calculated as follows:

$$O = \sum_{(n,g)\in\rho(G)} \binom{|C_g|}{n} = \sum_{(n,g)\in\rho(G)} \frac{|C_g|!}{n!\,(|C_g| - n)!}, \tag{6-5}$$

where $\rho(G)$ is a function to return a set of pairs that includes the number of groups n with the same group size in G and one of these groups $g \in G$. This O corresponds to the search space in brute-force search as one of indicators of the computational complexity. All groups with a same group size have always the same combinations of members. In its implementation, all C_g with the same group size can be handled efficiently by using same data.

6.2.3 Formulation for Quantum Annealing

A model based on the mathematic model described in the previous section can be formulated for quantum annealing by penalty method as follows:

$$\mathcal{H} = \mathcal{H}_{obj} + \lambda_m \mathcal{H}_m + \lambda_g \mathcal{H}_g, \tag{6-6}$$

$$\mathcal{H}_{obj} = \sum_{g\in G} \sum_{c\in C_g} \left(\left(\sum_{m\in M_c} D_{c,m} \right) * \prod_{m\in M_c} x_{g,m} \right), \tag{6-7}$$

$$\mathcal{H}_m = \sum_{m\in M} \left(1 - \sum_{g\in G} x_{g,m} \right)^2, \tag{6-8}$$

$$\mathcal{H}_g = \sum_{g\in G} \left(N_g - \sum_{m\in M} x_{g,m} \right)^2. \tag{6-9}$$

The parameters λ_m and λ_g in Eq. (6-6) are penalty coefficients to avoid the constraint violation. Eq. (6-7), (6-8) and (6-9) are corresponding to Eq. (6-1), (6-2) and (6-3)

respectively. Although quantum annealing solves a combinatorial optimization problem as QUBO in Eq. (2-3), our model is not QUBO but a higher order binary optimization (HOBO) model caused by Eq. (6-9). To solve our model on quantum annealing machine, we transform our HOBO model to a QUBO model by [123] that can realize the transformation with fewer slack variables by replacing a pair of decision variables consisting higher order and appearing more frequently with slack variables in priority. We use this "HOBO to QUBO" function implemented in Dimod higher order modules [124].

6.3 Evaluation

This section evaluates empirically the quantum annealing performance of the grouping optimization through our model formulated in Eq. (6-6), assuming in situation of collaborative learning that minimizes drop-out probability calculated by all members' personality compatibility. The evaluation uses three solvers and data for this evaluation. Then, after tuning of penalty coefficients in our model for these solvers, the performance of our model is evaluated.

6.3.1 Evaluation Settings

This evaluation compares quantum annealing with two solvers, simulated annealing and hill climbing, to evaluate our model for general grouping optimization. Evaluation environments of these solvers are described below.

Quantum Annealing (QA): we used D-Wave Advantage system 4.1 [103], which is a hardware realization of QA. The hardware settings are basically defaults. D-Wave Ocean SDK 4.4.0 [104] and JijModeling 0.9.14 [125] are used as software for the implementation of our model and the execution it on QA. The chain strength that avoids to break chains generated by minor embedding for Pegasus graph of qubits is automatically set through an embedding function in the Ocean SDK. A penalty parameter for "HOBO to QUBO" function in Dimod to avoid the constraint violations regarding slack variables is set to 2.4.

Simulated Annealing (SA): we used JijSAsampler implemented in JijZept 1.9.2 [126]. This executes SA on single thread not on parallel. The simulation parameters are default. Because JijZept can interpret a HOBO model internally without converting a

QUBO model, there is no penalty parameter for slack variables.

Hill Climbing (HC): we used an original implementation by R language. The process flow of HC for this problem are below:

1) Initialize a random solution.
2) Choose two members randomly, and swap them.
3) Apply this if improving the objective value.
4) Repeat 2) and 3) until all swap patterns aren't improved.

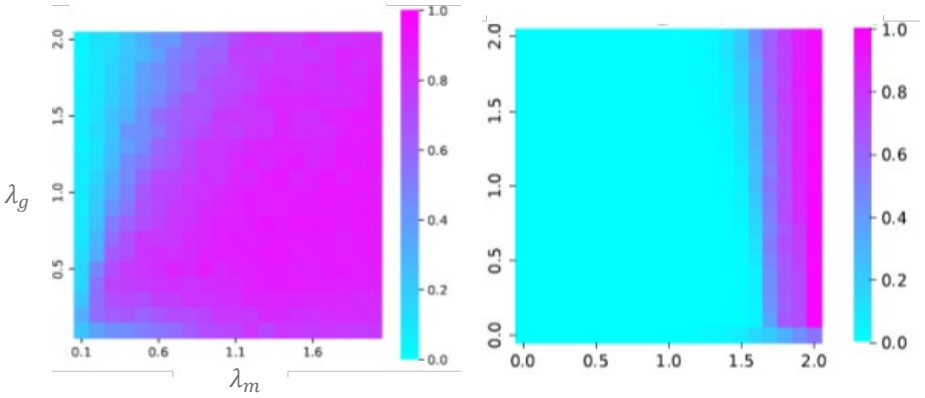
The evaluation data was prepared in pseudo-situation, assuming a collaborative learning. The data includes 9 members joining a learning course and divided to 3 groups. The indicator $D_{c,m}$ for the optimization is defined to drop-out probability calculated by all members' personality compatibility in each candidate group. In this data, the number of candidate groups is 84 from Eq. (6-4), and the number of all combination of this problem is 95,284 from Eq. (6-5). This data uses 75 qubits in QA machine in this evaluation, including 27 binary decision variables and 48 slack variables.

6.3.2 Tuning Penalty Coefficients

Before performance evaluation, penalty coefficients λ_m and λ_g of our model for the grouping optimization are determined in order to maximize the performance of QA and SA. In this parameter tuning, we measured the feasible solution rate and average objective value when these penalty coefficients are varied. Each run at particular penalty coefficients obtains 1,000 solutions by set the solver parameter *num_reads* to 1,000 in QA and SA. The feasible solution rate is the probability of obtaining a feasible solution that does not violate any constraints, neither Eq. (6-8) nor (6-9). This evaluation measured the feasible solution rate how many feasible solutions are obtained from 1,000 solutions in a single run. The average objective value is calculated by the average of objective function of Eq. (6-7) in feasible solutions obtained in a single run. The runs are with *annealing_time* set to 10 μs for QA and *num_sweeps* set to 1,000 for SA, as the solver parameters determined in the runtime.

Figure 6-2 shows the results of measuring feasible solution rates of our model as heatmaps by varying the penalty coefficients. In these heatmaps, the horizontal axis is λ_m, and the vertical axis is λ_g. A blue dot in the heatmap indicates a low feasible solution rate, while a purple dot indicates a high rate. Figure 6-2(a) and Figure 6-2(b) are results of QA and SA respectively to tune the penalty coefficients. QA obtains high feasible

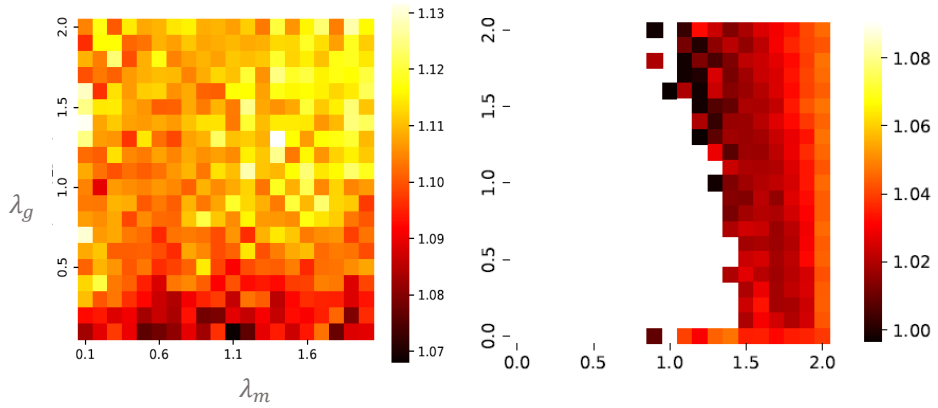
solution rates mostly when both λ_m and λ_g is more than 0.1 except for large λ_g and small λ_m. On the other hand, SA obtains high feasible solution rates only when λ_m is more than 1.6. In addition, λ_g is little relevant to the rate in SA.



(a) QA (b) SA

Figure 6-2: Heatmaps of feasible solution rates with varied penalty coefficients.

Next, based on the above results of feasible solution rate, the results of average objective values shown in Figure 6-3 is evaluated to determine the values of the two penalty coefficients. These heatmaps are similar to Figure 6-2, but the colors of heatmap are different. A yellow dot indicates a big average objective value, while a dark red dot indicates a small value that means a good solution. A white dot means no feasible solution. In QA shown in Figure 6-3(a), the smaller λ_g, the better solution is obtained. λ_m is also tends to decrease the average objective values, though not as much as λ_g. In SA shown in Figure 6-3(b), similar to feasible solution rates, the average objective values depend on only λ_m. The smaller λ_m, the better solutions are obtained. As these results, we found that better values of penalty coefficients differ between QA and SA.



(a) QA (b) SA

Figure 6-3: Heatmaps of average objective values with varied penalty coefficients.

6.3.3 Performance Evaluation

To evaluate the QA performance of the grouping optimization, we measured the objective value and TTS that is time to a feasible solution defined as below:

$$TTS = \tau \left\lceil \frac{\ln(1 - r_m)}{\ln(1 - r_1)} \right\rceil, \tag{6-10}$$

where r_1 is the probability of a feasible solution in a single run obtaining one solution, r_m is the probability of obtaining a feasible solution in m runs, and τ is the execution time per run. In this performance evaluation, we set r_m in TTS to 0.99 as a fixed value, and measured r_1 and τ in addition to the objective value by executing 1,000 times on each solver. In addition, for QA and SA, we measured them by varying execution time through these parameters, *annealing_time* and *num_sweeps*. The penalty coefficients of our models for QA are $\boldsymbol{\lambda_m}$=0.8 and $\boldsymbol{\lambda_g}$=0.5, and ones for SA are $\boldsymbol{\lambda_m}$=1.7, $\boldsymbol{\lambda_g}$=0.1.

Figure 6-4 shows the results of performance evaluation on the three solvers. QA with our model is the highest performance in TTS aspect. QA solves this problem 134 times faster than HC, and 392 times faster than SA. On the other hand, the objective values of QA are unstable even if the execution time is increased. This is assumed to be due to quantum errors occurring in the QA qubits [127].

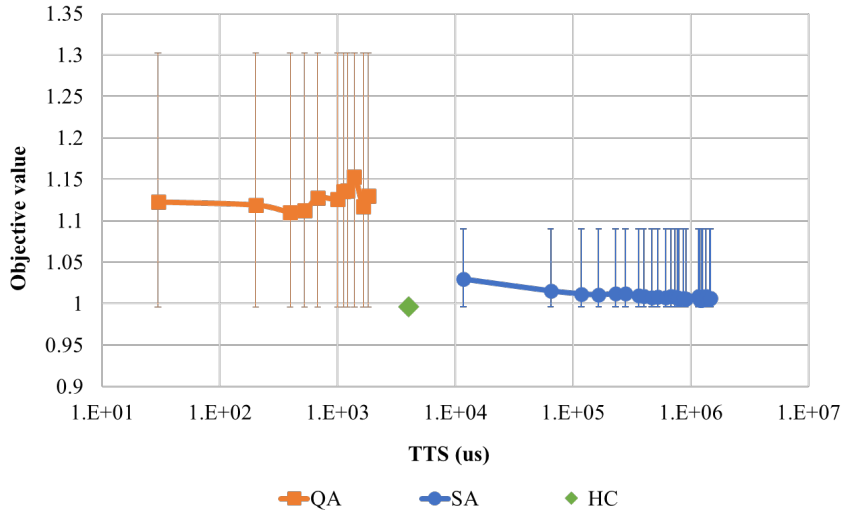


Figure 6-4: Performance comparison among QA, SA, and HC.

6.4 Summary

This chapter formulates a general model of the grouping optimization for quantum annealing, and evaluates it assuming collaborative learning on quantum annealing machine in comparison with simulated annealing and hill climbing. As a result, it is confirmed that quantum annealing solves this problem 134 times faster than the hill climbing and about 392 times faster than simulated annealing, although the stability of solution's accuracy on quantum annealing machine is lower than these methods. Evaluation on bigger group optimization problem is one of the important future works. However, current hardware of quantum annealing has a significant limitation of qubits. To expect the appearance of bigger hardware, we consider to evaluate the bigger problem on a quantum annealing simulator or Ising machine.

Chapter 7
Conclusion

In this thesis, we proposed a system architecture that achieves both faster data collection by data virtualization and faster solution derivation by quantum annealing as a data-driven system for solving combinatorial optimization problems. In addition, we improved the query performance in data virtualization and clarified the performance of quantum annealing in real problems in this system. This confirmed the realization of a system architecture that collects data with minimal overhead and performs optimization processing at high speed for combinatorial optimization problems.

This chapter concludes this thesis by discussing the end-to-end performance of the proposed system from the user's perspective and describing the remaining issues related to data virtualization and quantum annealing comprehensively including quantum computing, which are the technologies that comprise the data-driven system, as well as the system implementation of machine learning that is an important component of the proposed data-driven system.

This chapter is organized as follows: Section 7.1 discusses an end-to-end performance of the proposed data-driven system, Section 7.2 describes future direction with the remaining issues of the proposed data-driven system, and Section 7.3 concludes the thesis.

7.1 End-to-End Performance of our proposed data-driven system

The proposed data-driven system consists of both a data virtualization system that virtually integrates the data required to solve combinatorial optimization problems and a quantum annealing machine that actually solves the combinatorial optimization problems. In order to realize this data-driven system, this study resolved the respective

issues of query processing performance of the data virtualization system and practicality of the quantum annealing machine. As a result, compared to conventional data-driven systems, not only we improve the simplicity of environment construction, system operability, and the speed of data collection in the data platform, respectively, but we also made it possible to solve combinatorial optimization problems at high speed without compromising the speed of data utilization. On the other hand, the impact of the improved query processing performance in data virtualization on combinatorial optimization with quantum annealing has not been discussed. This section discusses the end-to-end performance on the entire data-driven system for VRP and grouping optimization, which are actual problems evaluated in our research on quantum annealing.

7.1.1 VRP

VRP is the problem of determining the most efficient order in which multiple vehicles can travel between all locations. As described in Chapter 5, quantum annealing can obtain a better solution than conventional methods on classical computer and has the potential to be superior in terms of speed at large scales. To solve VRP with quantum annealing, it is necessary to construct a QUBO, which requires travel costs between all locations. In the demand transportation service assumed as an application of VRP in Chapter 5, this travel cost between locations is the cost of traveling by bus, cab, or other vehicle between the pick-up and drop-off locations in the demand. The actual travel cost used in this problem is, for example, the traveling distance or time. This travel cost information depends on the available data and the accuracy of the traveling route required by the application.

In the evaluation in Chapter 5, the Euclidean distance calculated from the coordinates of the locations is used as the travel cost. In this case, the data to be obtained from the data platform is either the coordinate data of the locations or the distance data that has been previously calculated for all locations and stored. When the distance is calculated from the former coordinate data by SQL query, the proposed data-driven system is expected to work effectively due to benefit of query pushdown in data virtualization as proposed in Chapter 4.

If there is a need to calculate travel costs more accurately, travel time is used as a travel cost. Travel time can accurately represent actual travel cost, but it requires more data. Travel time between two locations can be calculated based on travel distance and travel

speed. The simplest way is to use Euclidean distance as the distance between two locations and the average speed of the vehicle as the travel speed. Here, the proposed data-driven system is superior in that it does not need to be aware of the location of the data if the data of the locations and the data of the vehicles are in different DBs. However, since the calculation of travel time between all locations is the outer product of the location data and the vehicle speed data, the methods in Chapter 3 and Chapter 4 are ineffective and the performance is comparable to that of running on an application server.

On actual roads, routes between locations are not always straight, so it is necessary to find the shortest route from multiple route candidates. In addition, the shortest route cannot be calculated based on distance alone because it is affected by various factors such as speed limits, traffic signals, and congestion on each road. Since some roads may be unavailable due to construction or other reasons, the latest road information is also necessary. Vehicle speeds also vary depending on road restrictions, weather conditions, and other factors. Therefore, a large amount of data and calculations are required to obtain accurate travel times. In combinatorial optimization problem that requires such multiple types of data, a data-driven system with data virtualization is effective in acquiring these data.

7.1.2 Grouping Optimization

The grouping optimization is the problem of maximizing the effectiveness of a grouping by composing groups; as described in Chapter 6, we showed that quantum annealing can find a solution faster than conventional methods, although the quality of the solution is unstable. In the optimization of the grouping optimization by quantum annealing, the data required for QUBO are the indicators of members in the candidate group. The sum of these indicators represents the effectiveness of the group.

In Chapter 6, as a grouping optimization for cooperative learning, the drop-out probability of a learning course was used as an indicator. This drop-out probability is calculated by a machine learning model that uses information on the attributes of the members, the educational course, and the personality compatibility among the members that make up the group as features [121]. In large companies that provide multiple services, it is common for customer information to be centralized, and for data on individual services, such as educational services, to be managed in separate DBSs. Therefore, information on member attributes and learning courses in this grouping

optimization may exist in different DBSs. In such cases, a data-driven system with data virtualization for data acquisition will be effective. In particular, for member attribute information, the query pushdown and intermediate data relocation methods in data virtualization described in Chapter 4 will be effective because only specific information on specific members receiving educational services will be acquired.

7.2 Future direction of this research

This study focused on improving the query processing performance of data virtualization and evaluating the utility of quantum annealing in the proposed data-driven system. On the other hand, there are still technical issues that need to be addressed to further improve the performance and usability of this system and to create many effective use cases. This section describes the remaining challenges for each of the data virtualization and quantum annealing technologies, as well as the challenges for the system technology targeting machine learning on application servers, which were not described in this study.

7.2.1 Data Virtualization

Data virtualization has the potential to realize data integration for various interfaces and data formats, not limited to RDBMS as a data source. There are various other implementations of database systems such as key-value databases, document-oriented databases, and so on. In particular, graph databases [128], which target data in graph form, have been developed for use in graph analysis, knowledge graph utilization, and graph machine learning. Because graph databases represent data as objects and relations, SQL queries in conventional RDBMSs cannot be used. Typical query languages include SPARQL [129] standardized by W3C and Cypher [130] developed for Neo4j [131], a typical graph database system. Interfaces in these languages different from SQL may cause a barrier to use for users who usually use SQL, and there is a demand to covert and integrate the interfaces. In addition, there is a high possibility of combining graph data with other data, such as machine learning that combines regular table format and graph data to create features, so it is meaningful to integrate graph databases through data virtualization. However, current data virtualization systems can only process data in table format. Data virtualization system requires a mechanism for acquiring and processing

graph data. A mechanism to decompose SQL queries and convert them to SPARQL or Cypher is also necessary.

Next, we discuss the data management perspective. Since data virtualization does not physically integrate databases into one database, these databases can be integrated without modifications. Therefore, administrators can build an integrated data environment more easily and quickly by utilizing data virtualization and its users can view a large number of data more easily and quickly than with physical integration. In this environment, users have knowledge of the data they have been using, but they do not have knowledge of other data that have been integrated. This creates a barrier to use data. As a mechanism to promote the use of such data, it is essential to introduce a system that allows users to browse and search the definitions and usage rules of the integrated data, called metadata. A database system for the metadata is called a data catalog, and collecting the metadata from data sources is called data profiling [132]. The data catalog can be divided into technical data and business data. The technical data can be collected directly from data sources by data profiling, such as data definitions, statistical data, and data provenance. The larger the size of managed data, the more time is required for its data profiling and the more difficult it is to maintain up-to-date technical data. This means high performance methods are required for the data profiling. The business data is information created by humans for its use, such as the operational meaning and usage rules of data. Business data must be input by users such as administrators and users, and automatic generation of business data to reduce the time and effort required is a major issue. Another important issue to promote the use of data virtualization is the realization of data governance to control the rules of data use, such as privacy preserving. While it is important to organize the rules from a legal perspective, it is also necessary to have the technology to strictly implement and control the rules. The elements and requirements of data management, including data governance, are described in detail in reference [133].

7.2.2 Quantum Annealing and Quantum Computing

The current implementation of quantum annealing is the D-Wave Advantage, which has about 5000 qubits [103]. The qubits in this implementation are connected by a special graph structure called the Pegasus graph, which limits the available interactions between the qubits. If the Ising model of the problem to be solved is not applicable to this graph,

multiple qubits are connected in a bound state called a chain to represent a single qubit. As described in Chapter 5 and Chapter 6, graph embedding techniques such as minor embedding are used for this purpose. In the case of full coupling using all J_{ij} in the Ising model, the number of decision variables that can be optimized with D-Wave Advantage is about 120 at most. However, the number of decision variables in a real-world problem would be well over several hundred. It is precisely for such large-scale problems that classical computers are difficult to run, and quantum annealing's high speed can be fully benefited. It is expected that the upper bound on the number of these variables will increase with the evolution of hardware in the future, but the timing and scale of this increase are unknown. Therefore, a technology is needed that allows large-scale problems to be solved in software with current hardware. Specifically, examples of such techniques include graph embedding, which reduces the number of qubits to be chained for each problem, and multiple runs on quantum annealing to solve a problem. Another approach to solve larger problems is to be hybrid with classical methods [134]; D-Wave offers a hybrid solver service that combines a quantum annealing machine with classical Tabu search [135]. Similarly, a hybrid solver combining a fast classical Ising machine is also possible.

To utilize quantum annealing, users are required to formulate a combinatorial optimization problem into a mathematical model and convert it into a QUBO. The conversion from a mathematical model to a QUBO can basically be done on a rule basis. QUBO formulas for general combinatorial optimization problems have also been proposed [102], so the problem to be solved can be mapped or extended to these formulas. On the other hand, these QUBO formulas are formulated in a naive way, and it may be possible to reduce the number of required decision variables by constructing them more efficiently. This may allow for larger scale problems to be handled. In addition, some real-world problems are so complex that they cannot be handled analytically, such as the objective function, making it difficult to construct QUBO. For such problems, a black-box optimization method [136] has been proposed that simulates the objective function for a certain solution and automatically generates and optimizes a QUBO. This method would enable the use of quantum annealing in complex combinatorial optimization problems using wireless simulators in the communications domain.

To use quantum annealing, the user's formulation of QUBO must be compiled into a physical model that can be executed by the machine. This includes the process of

expanding QUBO from an expression into a Hamiltonian matrix, converting it to an Ising model, and embedding the graph into the machine's qubit implementation. PyQUBO [137], for example, provides an interface to intuitively program QUBOs as mathematical expressions and automatically compile them into a model that can be executed by a quantum annealing machine. JijModeling provides a function to compile a mathematical model (objective function and constraint formulas) into QUBO [125]. Since the matrix representing QUBO is N^2 in size for N qubits, the processing load for its compilation cannot be ignored for the proposed system as a whole. Analysis of this processing load and its performance improvement are also important issues.

Although this paper focuses on quantum annealing, as discussed in Chapter 2, it is becoming possible to utilize more general-purpose gate-type quantum computers, and quantum algorithms capable of solving combinatorial optimization problems are also emerging. As mentioned in Chapter 6, quantum annealing tends to make the value of the objective functions unstable due to quantum errors. There are some researches to correct the quantum errors as post-processing of quantum annealing [138][139], but it is difficult for current quantum annealing machine, which is an analog quantum computer, to fundamentally be error tolerance. Although the quantum error occurs in current NISQ devices, there is potential for future improvement through quantum error correction as FTQC in addition to reducing quantum errors software-wise through quantum error mitigation [140]. However, very few studies have applied quantum algorithms for solving combinatorial optimization problems, such as QAOA and the Grover algorithm, to real problems and compared them with quantum annealing and classical methods. This means their effectiveness is still uncertain. Therefore, it is necessary to verify the effectiveness of various combinatorial optimization problems on gate-type quantum computers, including the problems evaluated in Chapter 5 and Chapter 6.

7.2.3 Machine Learning System

In solving the combinatorial optimization problem in the proposed data-driven system, machine learning may be used to generate the data necessary to construct QUBO, such as in the grouping problem in Chapter 6, where a machine learning model is used to calculate the drop-out probability of each learner in cooperative learning. The delivery planning problem in Chapter 5 may also use machine learning models to calculate the costs between locations, as described in Section 7.1.1. This paper does not describe the

system architecture for performing this machine learning, but in reality, many system components are required [141]. These system components realize the various steps of machine learning, called MLOps. These include the creation of the training data necessary for training phase, the management for the process of training machine learning models and verifying their performance, the deployment of trained models for use as inference services, and the performance monitoring and re-training of models used in services. There are various technological elements and tools to effectively and efficiently execute these processes [142]. It is necessary to consider how they should be integrated with the proposed data-driven system for solving combinatorial optimization problems.

7.3 Conclusion Remarks

The general flow of efforts to utilize data is as follows: analysis of the current status, predicting the future, and execution of measures, in that order. In the analysis of the current status, the past and current conditions are grasped from the data. This is done using data analysis based on conventional data mining techniques and machine learning for tasks such as recognition and classification. Predicting unknown matters, such as the future, uses machine learning based on the current situation. The results of these current state analyses and predictions are a kind of data, and how to effectively utilize such data is important. The final step is the execution of measures, and the utilization of data is considered to be completed when the optimal measures are selected according to the results of the analysis of current status and predicting the future. The proposed data-driven system for combinatorial optimization problems is one implementation of a system to realize this.

We believe that the utilization of data enriches people's lives. As a major step toward achieving this, we have proposed a data-driven system that can easily and quickly solve combinatorial optimization problems, which have been difficult to solve in the past. Although there are still challenges to be overcome in order to solve a large number of problems in a generalized manner, we believe that the development of the framework will lead to an evolution in this area.

References

[1] L. A. Wolsey and G. L. Nemhauser, “Integer and combinatorial optimization,” John Wiley & Sons Inc., 1999, ISBN: 978-0-471-35943-2.

[2] M. M. Flood, "The Traveling-Salesman Problem," Operations Research, Vol. 4, No. 1, pp.61-75, 1956, doi: 10.1287/opre.4.1.61.

[3] R. K. Ahuja, K. Mehlhorn, J. Orlin and R. E. Tarjan, "Faster algorithms for the shortest path problem," Journal of the ACM, Vol. 37, No. 2, pp 213–223, 1990, doi: 10.1145/77600.77615.

[4] T. R. Jensen and B. Toft, "Graph Coloring Problems," Wiley, 2011, ISBN: 978-1-118-03074-5.

[5] H. Kellerer, U. Pferschy and D. Pisinger, "Knapsack Problems," Springer, 2004, doi: 10.1007/978-3-540-24777-7.

[6] F. Glover and C. Mcmillan, "The general employee scheduling problem: An integration of MS and AI," Computers & operations research, Vol. 13, No. 5, pp. 563–573, 1986, doi: 10.1016/0305-0548(86)90050-X.

[7] C. H. Papadimitriou and K. Steiglitz. “Combinatorial optimization: algorithms and complexity,” Courier Corporation, 1998, ISBN: 978-0-486-40258-1.

[8] A. Doan, A. Halevy, and Z. Ives, “Principles of Data Integration, Morgan Kaufmann”, 2012, ISBN: 978-0124160446.

[9] S. R. Gardner, “Building the data warehouse,” Commun. ACM, 1998, doi: 10.1145/285070.285080.

[10] F. Nargesian et al., “Data lake management: challenges and opportunities,” Proc. VLDB Endow., 2019, doi: 10.14778/3352063.3352116.

[11] M. Kiran et al., “Lambda architecture for cost-effective batch and speed big data processing,” in IEEE Big Data, 2015, doi: 10.1109/BigData.2015.7364082.

[12] T. Achterberg and R. Wunderling, “Mixed Integer Programming: Analyzing 12 Years of Progress,” in Facets of Combinatorial Optimization, Springer Berlin

Heidelberg, pp. 449–481, 2013, doi: 10.1007/978-3-642-38189-8_18.

[13] A. Vince, “A framework for the greedy algorithm,” Discrete Applied Mathematics, vol. 121, no. 1–3, pp. 247–260, 2002, doi: 10.1016/S0166-218X(01)00362-6.

[14] E. Aarts and J. K. Lenstra, “Local Search in Combinatorial Optimization,” John Wiley & Sons Ltd., 1997, ISBN: 978-0-691-11522-1.

[15] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer, “Evolution algorithms in combinatorial optimization,” Parallel Computing, vol. 7, no. 1, pp. 65–85, 1988, doi: 10.1016/0167-8191(88)90098-1.

[16] R. S. Sutton and A. G. Barto, “Reinforcement learning: an introduction, Second edition,” The MIT Press, 2018, ISBN: 978-0-262-03924-6.

[17] C. A. Mack, “Fifty Years of Moore’s Law,” IEEE Transactions on Semiconductor Manufacturing, vol. 24, no. 2, pp. 202–207, 2011, doi: 10.1109/TSM.2010.2096437.

[18] Rick F. van der Lans, “Data Virtualization for Business Intelligence Systems,” Morgan Kaufmann, 2012, ISBN: 9780123944252.

[19] T. Kadowaki and H. Nishimori, “Quantum annealing in the transverse Ising model,” Phys. Rev. E - Stat. Physics, Plasmas, Fluids, Relat. Interdiscip Top., vol. 58, no. 5, pp. 5355–5363, 1998, doi: 10.1103/PhysRevE.58.5355.

[20] M. W. Johnson et al., “Quantum annealing with manufactured spins,” Nature, vol. 473, no. 7346, pp. 194–198, 2011, doi: 10.1038/nature10012.

[21] E. G. Rieffel et al., “A case study in programming a quantum annealer for hard operational planning problems,” Quantum Information Processing, vol. 14, no. 1, 2014, doi: 10.1007/s11128-014-0892-x.

[22] N. Hamada, K. Saito, and H. Kawashima, “Applying Quantum Annealing for Shift Scheduling Problem for Call Centers,” International Journal of Networking and Computing, vol. 13, no. 1, pp. 2–17, 2023, doi: 10.15803/ijnc.13.1_2.

[23] F. Phillipson and H.S. Bhatia, “Portfolio Optimisation Using the D-Wave Quantum Annealer,” Computational Science – ICCS 2021, Springer, 2021, doi: 10.1007/978-3-030-77980-1_4

[24] K. Saito, A. Ikami, and C. Ono, “Evaluating dynamic spectrum allocation using quantum annealing,” IEICE Communication Express, vol. 10, no. 9, pp. 726–732, 2021, doi: 10.1587/comex.2021XBL0047.

[25] L. K. Grover, “A fast quantum mechanical algorithm for database search,” in Proc.

of 28th Annual Symposium on the Theory of Computing (STOC), pp. 212-219, 1996, doi: 10.1145/237814.237866.

[26] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," SIAM Review, vol. 41, no. 2, pp. 303-332, 1999, doi: 10.1137/S0036144598347011.

[27] D. Coppersmith, "An approximate Fourier transform useful in quantum factoring," Technical Report RC19642, IBM, 1994, doi: 10.48550/arXiv.quant-ph/0201067.

[28] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for linear systems of equations," Physical Review Letter, vol. 103, no. 150502, 2009, doi: 10.1103/PhysRevLett.103.150502.

[29] IBM Quantum, https://www.ibm.com/quantum

[30] F. Arute, et al., "Quantum supremacy using a programmable superconducting processor," Nature, vol 574, pp. 505-510, 2019.

[31] J. Preskill, "Quantum computing in the NISQ era and beyond," Quantum Journal, vol. 2, no. 79, 2018, doi: 10.22331/q-2018-08-06-79.

[32] K. Mitarai et al., "Quantum Circuit Learning," Physical Review A, vol. 98, no. 032309, 2018, doi: 10.1103/PhysRevA.98.032309.

[33] A. Peruzzo et al., "A variational eigenvalue solver on a photonic quantum processor," Nature Communication, vol. 5, no. 4213, 2014, doi: 10.1038/ncomms5213.

[34] E. Farhi, J. Goldstone, and S. Gutmann, "A Quantum Approximate Optimization Algorithm," arXiv:1411.4028, 2014, doi: 10.48550/arXiv.1411.4028.

[35] T. Hogg and D. Portnov, "Quantum optimization," Information Sciences, vol. 128, no. 3-4, pp. 181-197, 2000, doi: 10.1016/S0020-0255(00)00052-9.

[36] C. A. Trugenberger, "Quantum optimization for combinatorial searches," New Journal of Physics, vol. 4, No. 26, 2002, doi: 10.1088/1367-2630/4/1/326

[37] J. Bang et al., "Quantum heuristic algorithm for traveling salesman problem," Journal of the Korean Physical Society, vol. 61, no. 12, pp. 1944–1949, 2012, doi: 10.3938/jkps.61.1944.

[38] A. Callison et al., "Finding spin glass ground states using quantum walks," New Journal of Physics, vol. 21, no. 12, p. 123022, 2019, doi: 10.1088/1367-2630/ab5ca2.

[39] N. Slate, et al, "Quantum walk-based portfolio optimisation," Quantum Journal,

vol. 5, p. 513, 2021, doi: 10.22331/q-2021-07-28-513.

[40] M. T. Özusu and P. Valduriez, "Principles of Distributed Database Systems," Third Edition, Springer, 2011, ISBN: 978-3-030-26252-5.

[41] Apache Hadoop, http://hadoop.apache.org/

[42] Apache Hive, http://hive.apache.org/

[43] Impala, http://impala.io/

[44] Apache Drill, http://drill.apache.org/

[45] Presto, http://prestodb.io/

[46] Apache HBase, https://hbase.apache.org/

[47] Apache Kudu, https://kudu.apache.org/

[48] Apache Pig, https://pig.apache.org/

[49] Apache Solr, http://lucene.apache.org/solr/

[50] Apache Giraph, https://giraph.apache.org/

[51] V. S. Subrahmanjan, et al., "HERMES: A Heterogeneous Reasoning and Mediator System," Available at http://www.cs.umd.edu/projects/hermes/overview/paper, 1995.

[52] R. Ahmed, et al., "The Pegasus Heterogeneous Multidatabase System. IEEE Computer," Vol. 24, No. 12, pp. 19–27, 1991, doi: 10.1109/2.116885.

[53] S. Chawathe, et al., "The TSIMMIS Project: Integration of Heterogeneous Information Sources," Journals of Intelligent Information System, Vol. 8, No. 2, pp. 117–132, 1994.

[54] M. J. Carey, et al., "Towards heterogeneous multimedia information systems: the Garlic approach," in Proceedings of RIDE-DOM, pp. 124–131, 1995, doi: 10.1109/RIDE.1995.378736.

[55] W. Litwin, et al., "Interoperability of Multiple Autonomous Databases. ACM Computing Surveys," Vol. 22, No. 3, pp. 267–293, 1990, doi: 10.1145/96602.96608.

[56] A. P. Sheth and J. A. Larson, "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases," ACM Computing Surveys, Vol. 22, No. 3, pp. 183–236, 1990, doi: 10.1145/96602.96604.

[57] S. Fiore, et al., "Data Virtualization in Grid Environments through the GReIC Data Access and Integration Service," In Proceedings of ICITST, pp. 1–6, 2009,

doi: 10.1109/ICITST.2009.5402520.

[58] M. Salloum, et al., “Online Ordering of Overlapping Data Sources," PVLDB, Vol. 7, No. 3, pp. 133-144, 2013, doi: 10.14778/2732232.2732233.

[59] M. Karpathiotakis, et al., “Just-In-Time Data Virtualization: Lightweight Data Management with ViDa,” In Proceedings of CIDR, pp. 1-11, 2015.

[60] Teiid: JBoss Project, http://teiid.jboss.org/

[61] M. Lenzerini, “Data Integration: A Theoretical Perspective,” In Proceedings of PODS, pp. 233-246, 2002, doi: 10.1145/543613.543644.

[62] A. Abouzeid and K. Bajda-pawlikowski, “HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads,” PVLDB, Vol. 2, No. 1, pp. 922–933, 2009, doi: 10.14778/1687627.1687731.

[63] D. J. DeWitt, et al., “Split query processing in polybase,” in Proceedings of SIGMOD, pp. 1255–1266, 2013, doi: 10.1145/2463676.2463709. H. Lim, et al., “How to Fit when No One Size Fits,” in CIDR, 2013.

[64] H. Lim, et al., “How to Fit when No One Size Fits,” in CIDR, 2013.

[65] A. Simitsis, et al., “HFMS: Managing the lifecycle and complexity of hybrid analytic data flows,” in Proceedings of ICDE, pp. 1174–1185, 2013.

[66] H. Herodotou, et al., “Starfish: A Self-tuning System for Big Data Analytics,” in CIDR, pp. 261–272, 2011, doi: 10.1109/ICDE.2013.6544907.

[67] V. Borkar, et al., “Hyracks: A flexible and extensible foundation for data-intensive computing,” in ICDE, pp. 1151–1162, 2011, doi: 10.1109/ICDE.2011.5767921.

[68] TPC-H, http://www.tpc.org/tpch/

[69] TPC-H configured for Impala, https://github.com/kj-ki/tpc-h-impala

[70] N. Bruno and S. Chaudhuri, “Exploiting Statistics on Query Expressions for Optimization,” in Proceedings of SIGMOD, pp.263-274, 2002, doi: 10.1109/ICDE.2002.994703.

[71] N. Kabra and D. DeWitt, “Efficient mid-query re-optimization of sub-optimal query execution plans,” in Proceedings of SIGMOD, pp. 106–117, 1998, doi: 10.1145/276304.276315.

[72] D. Kossmann, M. J. Franklin and G. Drasch, “Cache Investment: Integrating Query Optimization and Distributed Data Placement,” ACM Transactions on Database Systems, Vol. 25, No. 4, 517–558, 2001, doi: 10.1145/377674.377677.

[73] S. Iyer, and A. Cheung, “Summarizing Source Code using a Neural Attention

Model," in Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pp. 2073–2083, 2016, doi: 10.18653/v1/P16-1195.

[74] P. Yu, and M. Chen, On Workload Characterization of Relational Database Environments," IEEE Transactions on Software Engineering, vol. 18, no. 4, pp. 347–355, 1992, doi: 10.1109/32.129222.

[75] Q. Yao, A. An, and X. Huang, "Mining and Modeling Database User Access Patterns," in Proceedings of the 16th International Symposium on Methodologies for Intelligent Systems, LNAI 4203, pp. 493–503 (2006), doi: 10.1007/11875604_56.

[76] H. Elmeleegy, A. Elmagarmid, and J. Lee, "Leveraging Query Logs for Schema Mapping Generation in U-MAP," in Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, pp. 121-132 (2011), doi: 10.1145/1989323.1989337.

[77] A. Ganapathi, H. Kuno, U. Dayal, et al., "Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning," in Proceedings of the IEEE 25th International Conference on Data Engineering, pp. 592–603, 2009, doi: 10.1109/ICDE.2009.130.

[78] J. Celko, "Trees and Hierarchies in SQL for Smarties. Second Edition," Morgan Kaufmann, 2012, ISBN: 978-0123877338.

[79] Vertica, https://www.vertica.com/

[80] PostgreSQL FDW, https://www.postgresql.org/docs/current/static/postgres-fdw.html

[81] Oracle Database Link, http://docs.oracle.com/database/122/ADMIN/distributed-database-concepts.htm#ADMIN12084

[82] A. P. Sheth, and J. A. Larson, "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases," ACM Computing Surveys, vol. 22, no. 3, pp. 183–236, 1990, doi: 10.1145/96602.96604.

[83] R. Ramakrishnan, and J. Gehrke, "Database Management Systems, Third Edition," McGraw-Hill (2003), ISBN: 978-0071231510.

[84] L. M. Haas, D. Kossmann, E. Wimmers, et al., "Optimizing Queries across Diverse Data Sources," in Proceedings of the 23rd International Conference on Very Large Data Bases, pp. 276-285, 1997, doi: 10.5555/645923.670995.

[85] M. T. Ozsu, and P. Valduriez, "Principles of Distributed Database Systems, Third

Edition," Springer, 2011, ISBN: 978-1-4419-8833-1.

[86] S. Nural, P. Koksal, F. Ozcan, et al., "Query Decomposition, Optimization and Processing in Multidatabase Systems," in Proceedings of Symposium of the European Joint Conference on Engineering Systems Design and Analysis, pp. 41-52, 1996.

[87] Z. Li, and K. A. Ross, "PERF Join: An Alternative to Two-way Semijoin and Bloomjoin," in Proceedings of the Fourth International Conference on Information and Knowledge Management, pp. 137–144, 1995, doi: 10.1145/221270.221360.

[88] O. Polychroniou, R. Sen, and K. A. Ross, "Track join: distributed joins with minimal network traffic," in Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, pp. 1483–1494, 2014, doi: 10.1145/2588555.2610521.

[89] D. Kossmann, and K. Stocker, "Iterative Dynamic Programming: A New Class of Query Optimization Algorithms," ACM Transactions on Database Systems, Vol. 25, No. 1, pp. 43-82, 2000, doi: 10.1145/352958.352982.

[90] D. DeHaan, and F. W. Tompa, "Optimal Top-down Join Enumeration," in Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, pp. 785-796, 2007, doi: 10.1145/1247480.1247567.

[91] P. Fender, and G. Moerkotte, "Counter Strike: Generic Top-Down Join Enumeration for Hypergraphs," in Proceedings of the VLDB Endowment, Vol. 6, No. 14, pp. 1822–1833, 2013, doi: 10.14778/2556549.2556565.

[92] G. Moerkotte, and T. Neumann, Analysis of Two Existing and One New Dynamic Programming Algorithm for the Generation of Optimal Bushy Join Trees without Cross Products," in Proceedings of the 32nd International Conference on Very large Data Bases, pp. 930–941, 2006, doi: 10.5555/1182635.1164207.

[93] N. Bruno, and S. Chaudhuri, "Exploiting Statistics on Query Expressions for Optimization," in Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, pp. 263–274, 2002, doi: 10.1145/564691.564722.

[94] L. L. Perez, and C. M. Jermaine, "History-aware Query Optimization with Materialized Intermediate Views," in Proceedings of the IEEE 30th International Conference on Data Engineering, pp. 520–531, 2014, doi:

10.1109/ICDE.2014.6816678.

[95] D. Makreshanski, G. Giannikis, G. Alonso, et al., “MQJoin: Efficient Shared Execution of Main-Memory Joins,” in Proceedings of the VLDB Endowment, Vol. 9, No. 6, 480–491, 2016, doi: 10.14778/2904121.2904124.

[96] H. Lane, G. R. Thompson, C.-W. Chung, E. Barkmeyer, et al., “Heterogeneous Distributed Database Systems for Production Use,” ACM Computing Surveys, Vol. 22, No. 3, pp. 237–266, 1990, doi: 10.1145/96602.96607.

[97] G. B. Dantzig and J. H. Ramser, “The Truck Dispatching Problem,” Management Science, vol. 6, no. 1, pp. 80–91, 1959, doi: 10.1287/mnsc.6.1.80.

[98] K. Tsubouchi, H. Yamato, and K. Hiekata, “Innovative on-demand bus system in Japan,” IET Intelligent Transport Systems, vol. 4, no. 4, 2010, doi: 10.1049/iet-its.2009.0113.

[99] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, “On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment,” in Proceedings of the National Academy of Sciences, vol. 114, no. 3, pp. 462–467, 2017, doi: 10.1073/pnas.1611675114.

[100] V. Choi, “Minor-embedding in adiabatic quantum computation: I. the parameter setting problem,” Quantum Inf. Process., vol. 7, no. 5, pp. 193–209, 2008, doi: 10.1007/s11128-008-0082-9.

[101] Z. Bian, F. Chudak, R. Israel, B. Lackey, W. G. Macready, and A. Roy, “Discrete optimization using quantum annealing on sparse Ising models,” Front. Phys., vol. 2, no. 56, pp. 1–10, 2014, doi: 10.3389/fphy.2014.00056.

[102] A. Lucas, “Ising formulations of many NP problems,” Front. Phys., vol. 2, no. 5, pp. 1–14, 2014, doi: 10.3389/fphy.2014.00005.

[103] D-Wave Systems, https://www.dwavesys.com/solutions-and-products/systems/.

[104] D-Wave Ocean SDK, https://github.com/dwavesystems/dwave-ocean-sdk.

[105] Fujitsu Digital Annealer, https://www.fujitsu.com/jp/digitalannealer/.

[106] OpenJij, https://github.com/OpenJij/OpenJij

[107] S. Tsukamoto, M. Takatsu, S. Matsubara, and H. Tamura, “An accelerator architecture for combinatorial optimization problems,” Fujitsu Sci. Tech. J., vol. 53, no. 5, pp. 8–13, 2017.

[108] Google OR-Tools, https://developers.google.com/optimization.

[109] D. Pecin, A. Pessoa, M. Poggi, E. Uchoa, “Improved branch-cut-and-price for

capacitated vehicle routing," Mathematical programming Computation, vol. 9, no. 1, pp. 61-100, 2017, doi: 10.1007/s12532-016-0108-8.

[110] M. A. Mohammed, M. K. A. Ghanil, "Solving Vehicle Routing Problem by Using Improved K-Nearest Neighbor Algorithm for Best Solution," J. Comput. Sci., vol. 21, pp. 232-240, 2017, doi: 10.1016/j.jocs.2017.04.012.

[111] D. Pecin, C. Contardo, G. Desaulniers, Eduardo Uchoa, "New enhancements for the exact solution of the vehicle routing problem with time windows," INFORMS Journal on Computing, vol. 29, no. 3, pp. 489-502, 2017, doi: 10.5555/3215328.3215335.

[112] S. Ropke, D. Pisinger, "An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows," Transportation Science, vol. 40, no. 4, pp. 455-472, 2006, doi: 10.1287/trsc.1050.0135.

[113] T. T. Tran et al., "A hybrid quantum-classical approach to solving scheduling problems," in Proceedings of the 9th Annual Symposium on Combinatorial Search, pp. 98–106, 2016, doi: 10.1609/socs.v7i1.18390.

[114] V. N. Smelyanskiy et al., "A Near-Term Quantum Computing Approach for Hard Computational Problems in Space Exploration," arXiv:1204.2821, 2012, doi: 10.48550/arXiv.1204.2821.

[115] S. Feld et al., "A hybrid solution method for the capacitated vehicle routing problem using a quantum annealer," Front. ICT, vol. 6, no. 13, pp. 1-13, 2019, doi: 10.3389/fict.2019.00013.

[116] H. Irie, G. Wongpaisarnsin, M. Terabe, A. Miki, and S. Taguchi, "Quantum Annealing of Vehicle Routing Problem with Time, State and Capacity," in International Workshop on Quantum Technology and Optimization Problems, pp. 145–156, 2019, doi: 10.48550/arXiv.1903.06322.

[117] A. S. Goodsell, M. R. Maher, V. Tinto, B. L. Smith and J. MacGregor, "Collaborative learning: A sourcebook for higher education", Natl Center on Postsecondary, 1992, ISBN: 978-9994081820.

[118] Lewis R. Goldberg, "The Development of Markers for the Big Five Factor Structure," Psychological Assessment, vol 4, no. 1, pp.26–42, 1992, doi: 10.1037/1040-3590.4.1.26.

[119] T. Halfhill, E. Sundstrom, J. Lahner, W. Calderone, and T. M. Nielsen, "Group personality composition and group effectiveness an integrative review of

empirical research," Small Gr. Res., vol. 36, no. 1, pp. 83–105, 2005, doi: 10.1177/1046496404268538.

[120] M. A. G. Peeters, H. F. J. M. Van Tuijl, C. G. Rutte, and I. M. M. J. Reymen, "Personality and team performance: A meta-analysis," Eur. J. Pers., vol. 20, no. 5, pp. 377–396, 2006, doi: 10.1002/per.588.

[121] A. Kobayashi, Y. Ishikawa, K. Ikeda, D. Kamisaka, and R. Legaspi, "Composing Groups in Collaborative Learning by Pair Personality Differences," in Proceedings of the 31st ACM Conference on User Modeling, Adaptation and Personalization, pp. 116–123, 2023, doi: 10.1145/3565472.3592954.

[122] M. Ohzeki, Groupinq, Qubits 2021 Japan Conference, 2021, https://www.youtube.com/watch?v=Pu7qtRP5O2g.

[123] A. Mandal, A. Roy, S. Upadhyay, and H. Ushijima-Mwesigwa, "Compressed quadratization of higher order binary optimization problems," in Proceedings of the 17th ACM International Conference on Computing Frontiers, pp. 126–131, 2020, doi: 10.1145/3387902.3392627.

[124] Dimod, Higher-Order Models, https://docs.ocean.dwavesys.com/en/stable/docs_dimod/reference/models.html#higher-order-models

[125] JijModeling, https://www.jijmodeling.jijzept.com/.

[126] JijZept, https://www.jijzept.com/#/.

[127] T. Zaborniak and R. De Sousa, "Benchmarking Hamiltonian Noise in the D-Wave Quantum Annealer," IEEE Trans. Quantum Eng., vol. 2, pp. 1–6, 2021, doi: 10.1109/TQE.2021.3050449.

[128] I. Robinson, J. Webber, and E. Eifrem, "Graph Databases: New Opportunities for Connected Data," O'Reilly, 2015, ISBN: 978-1491930892.

[129] SPARQL, W3C, https://www.w3.org/TR/sparql11-overview/

[130] Cypher, https://neo4j.com/developer/cypher/

[131] Neo4j, https://neo4j.com/

[132] Z. Abedjan, L. Golab, and F. Naumann, "Profiling relational data: a survey," VLDB Journal, vol. 24, no. 4, pp. 557–581, 2015, doi: 10.1007/s00778-015-0389-y.

[133] DAMA-DMBOK2: Data Management Body of Knowledge, 2nd edition,

https://technicspub.com/dmbok2/

[134] A. Callison and N. Chancellor, “Hybrid quantum-classical algorithms in the noisy intermediate-scale quantum era and beyond,” Phys. Rev. A, vol. 106, no. 1, p. 010101, Jul. 2022, doi: 10.1103/PhysRevA.106.010101.

[135] “D-Wave Hybrid Solver Service: An Overview,” D-Wave Whitepaper Series, 14-1039A-B, 2020, https://www.dwavesys.com/media/4bnpi53x/14-1039a-b_d-wave_hybrid_solver_service_an_overview.pdf

[136] S. Izawa, K. Kitai, S. Tanaka, R. Tamura, and K. Tsuda, “Continuous black-box optimization with an Ising machine and random subspace coding,” Phys. Rev. Research 4, 023062, 2022, doi: 10.1103/PhysRevResearch.4.023062.

[137] PyQUBO, https://pyqubo.readthedocs.io/en/latest/

[138] K. L. Pudenz, T. Albash, and D. A. Lidar, “Error-corrected quantum annealing with hundreds of qubits,” Nature Communications, vol. 5, no. 1, pp. 3243, 2014, doi: 10.1038/ncomms4243.

[139] T. Śmierzchalski, Ł. Pawela, Z. Puchała, T. Trzciński, and B. Gardas, “Post-Error Correction for Quantum Annealing Processor using Reinforcement Learning,” arXiv:2203.02030, 2022, doi: 10.48550/arXiv.2203.02030

[140] Z. Cai et al., “Quantum Error Mitigation,” arXiv:2210.00921, 2023, doi: 10.48550/arXiv.2210.00921.

[141] D. Kreuzberger, N. Kühl, and S. Hirschl, “Machine Learning Operations (MLOps): Overview, Definition, and Architecture,” IEEE Access, vol. 11, pp. 31866–31879, 2023, doi: 10.1109/ACCESS.2023.3262138.

[142] N. Hewage and D. Meedeniya, “Machine Learning Operations: A Survey on MLOps Tool Support,” arXiv:2202.10169, 2022, doi: 10.48550/arXiv.2202.10169.