uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Li Zou

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M. (Computer Science)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Global Aggregate View Selection in a Peer Warehousing System

TITRE DE LA THÈSE / TITLE OF THESIS

Iluju Kiringa

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Liam Peyton

Michael Weiss

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Global Aggregate View Selection in a Peer Warehousing System

by

Li Zou
Supervised by Dr. Iluju Kiringa

A thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of
Master in Computer Science

Ottawa-Carleton Institute for Computer Science
School of Information Technology and Engineering
University of Ottawa

Your file Votre référence
ISBN: 978-0-494-49306-9
Our file Notre référence
ISBN: 978-0-494-49306-9

Abstract

Selecting views to materialize is one of the most important decisions to make when designing a data warehouse. In a peer data warehouse design, this problem is more difficult than in a centralized data warehouse design since the costs of global communication, data transfer, and data transformation between peers need to be additionally considered for getting the final integrated global query answers. The objective of our work is to select optimal sets of materialized aggregate views on different peers in a peer data warehousing system such that the total cost of answering the global queries posted on the given peer and maintaining the materialized views is minimized.

In this thesis, we develop a theoretical framework for analyzing and solving the P2P global materialized view (GMV) selection problem. We extend the concepts of *Expression AND-DAG*, *query aggregation lattice*, and cost model defined on *centralized data warehouses* to the peer data warehousing semantics. In our problem scope, P2P *Expression AND-DAG* and *P2P query aggregation lattice* are constructed dynamically. In our cost model, we take the data transfer costs between peers and global materialized view maintenance costs into consideration. Then we extend an existing centralized view selection greedy algorithm to solve our P2P view selection problem. We assume that peer data warehouse dimensions are consistent throughout the whole system. We also assume that there is only one instance of the P2P GMV selection algorithm executing in the P2P system. Finally, we simulate the P2P view selection algorithm. The simulation experiment results show that the query frequencies and the update frequencies of materialized views, as well as the data transfer rate have critical impacts on the final view selection results. Moreover, the longest path of the P2P network, the number of granularity levels in the global dimension lattice, and the number of data warehouse dimensions affect the P2P view selection algorithm processing time and the minimum combination cost of answering global queries and maintaining the materialized views.

Acknowledgements

I would like to thank my supervisor Dr. Iluju Kiringa. I am grateful to him for all his generous support and his patient guidance. He is an outstanding supervisor and mentor. I admire his solid knowledge and sincere research attitude. It is indeed my pleasure to have the opportunity to work with him.

I also thank Ruogu Luo for his help in implementing and testing my algorithm. Special thanks to my friend, Yi Zhang, and my colleague, Nicholas Marotta, for the time that they spent on reviewing my thesis.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

According to Bill Inmon [Chaudhuri1997, Inmon2001], a *data warehouse* is a *subject oriented*, *integrated*, *time variant*, *nonvolatile* collection of data in support of business management decisions. In today's competitive global business environments, understanding and managing enterprise-wide information is crucial for making timely decisions and responding to changing business conditions [Moeller2001].

In data warehouse environments, most decision-support queries are aggregate queries. The execution of aggregate queries is usually time consuming since computing one aggregate query often requires scanning much data, and a data warehouse usually contains enormous amount of data. Therefore, realizing efficient query processing is a critical issue in data warehouse environments. There are two popular approaches to improve the data warehouse system performance: optimizing aggregate queries and using materialized views. Materializing frequently asked aggregate queries is a very common technique to improve the system performance. So far much database research has been done on materialized views for centralized data warehouse systems. Especially, the Stanford Database Group has pioneered work in this area and made substantial contributions on materialized view management. A member of this group, Gupta, proposed a fundamental theoretical framework for materialized view selection [Gupta1997, Gupta1999] in a centralized data warehouse. We will elaborate more about this in Section 2.1.

In this chapter, the motivation behind our thesis is addressed first. We then describe the problem to be solved in this thesis, which is followed by a running example used throughout the thesis. Finally, the contributions and the outline of the thesis are presented.

1.1 Motivation

Espil et al. [Espil2004] have extended the traditional centralized data warehouse technology to a flexible and cooperative P2P architecture. In such a system, called a peer data warehouse system, the peer receiving queries is considered as *local data warehouse*, while

the other peers relevant to the local data warehouse are denoted as *acquaintances* or *relevant peers* of the local data warehouse. We will give a detailed definition of the notion of *relevant peer* in Section 2.4. Cooperative data exchange between peers is needed to answer the queries posed on the local data warehouse. Figure 1.1 shows how a global query is evaluated in a peer data warehousing system.

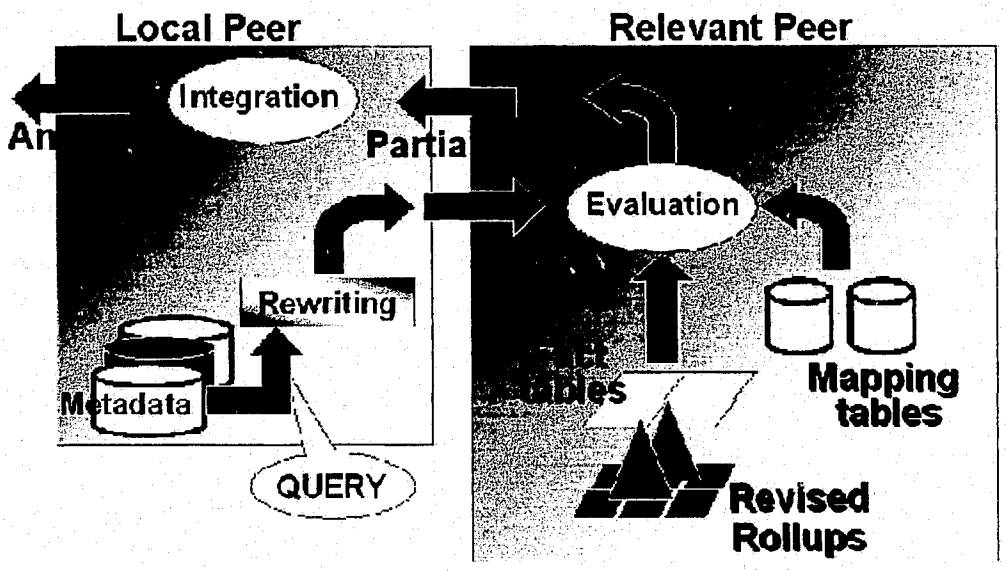


Figure 1.1 The query evaluation process in a peer data warehousing system

Furthermore, Espil et al. proposed a model for multidimensional data distributed in a P2P network, called a *P2P multidimensional data model*, to handle the data mapping and transformation between peers and evaluate multidimensional aggregation queries globally. In that model, each peer holds only the dimensions of interest at that peer; the fact table is structured from the bottom levels of the local dimensions in each peer and populated locally. Dimensions with the same semantics in different peers are called *peer dimensions*. Since the schemas of the local data warehouse and its acquaintances may be different, the mapping between peer dimensions in both schema and instance level is required. Accordingly, Espil et al. have proposed a revise and map approach to map and transform the aggregated data between the data requesting peer and its acquaintances. By using this approach, the instance of the acquaintance dimension will be revised to adapt it to the meaning of the local

dimension instance. Within the P2P data warehouse system context, each *peer fact table* is a specialization of the *generic fact table* appearing in the query. We will give the detailed explanations of the concepts of *peer fact table* and *generic fact table* in the coming section.

In general, the work of Espil et al. mainly focuses on how to integrate the data between peers by using the revise and map approach to integrate the dimensions under the common view of the local peer and consolidate the facts between peers by using the idea of considering each fact table in peers as the specialization of the generic fact table appearing in the queries. Deciding on how to store and maintain materialized views of aggregated data in different peers to improve the system performance and response time in a peer data warehouse system is one of the open problems mentioned in [Espil2004]. To the best of our knowledge, this problem still remains open.

Selecting views to materialize is one of the most important decisions to make when designing a data warehouse. A substantial amount of work has been done with regard to the materialized view selection problem in a *centralized data warehouse*. We will review the important related work in Section 2.1 in details. The data placement problem in P2P database domain is known as a complex and challenging research topic; the view selection problem in peer data warehouse systems is an interesting instance of data placement problem in the peer databases field [Gribble2001]. The objective of our work is to explore and provide a solution to the view selection problem in a peer warehousing system. Our solution will also solve the open problem mentioned above. In the next section, we describe the view selection problem in a peer data warehouse in details.

1.2 The Problem

Materialized view selection is one of the most important decisions to make in designing a data warehouse. In a peer data warehouse design, this problem is more difficult than in a centralized data warehouse design since the costs of global communication, data transfer, and data transformation between peers need to be additionally considered for getting the final integrated answers for global queries. When some views are materialized at the proper

locations in the P2P distributed network, the average query response time of the whole system will be greatly reduced since the data transfer or/and data transformation costs between the peers are eliminated. However, in this case instead we have to consider the global view maintenance cost, which is the cost to maintain the global view consistent with its data sources. Hence, the problem we want to solve is to select sets of materialized aggregate views on different peers to speed up the global queries posted on the given peer in a peer data warehousing system and minimize the total cost of answering queries and maintaining the selected materialized views.

To differentiate the view selection problem in a peer data warehousing system from that in a centralized data warehouse, we define the materialized view selection problem in the P2P architecture as *Global Materialized View* (GMV) *Selection Problem*, while we refer to the view selection problem in one data warehouse as to the *Local Materialized View* (LMV) *Selection Problem*. In this thesis, we are only concerned with the GMV selection problem.

1.3 Running Example

In this section, we first introduce a running example; and then we use it to illustrate how global materialized views speed up the global queries posted on a peer data warehousing system.

Consider an automobile company like General Motors. The headquarter of the company is located in the United States (US). There are four international branches distributed in the following four places in the world: Canada, China, Japan, and Korea. This company sells various categories of automobiles, such as vans, heavy trucks, motorcycles, and cars. Each category of vehicle has different brands and different models. Each site has a local data warehouse, which has its own technology, its own data, and so forth. Each data warehouse serves the same function, except that the scope of data warehouse is local. These local data warehouses are distributed in a peer-to-peer (P2P) network and must cooperate together to

provide the corporation analysis globally or partially. A simple topology of the peer-to-peer data warehouses system is shown in Figure 1.2.

The data warehouses use semantically similar schemas, but some dimensions in one data warehouse may not appear in other data warehouses, and the dimension hierarchies may be different from each other, though related. A sample schema of one local data warehouse is shown in Figure 1.3. In this schema, there are three hierarchy dimensions: Product, Time, and Location, and a base fact table, which is Sales. We will get back to details of Figure 1.3 in Section 2.2.

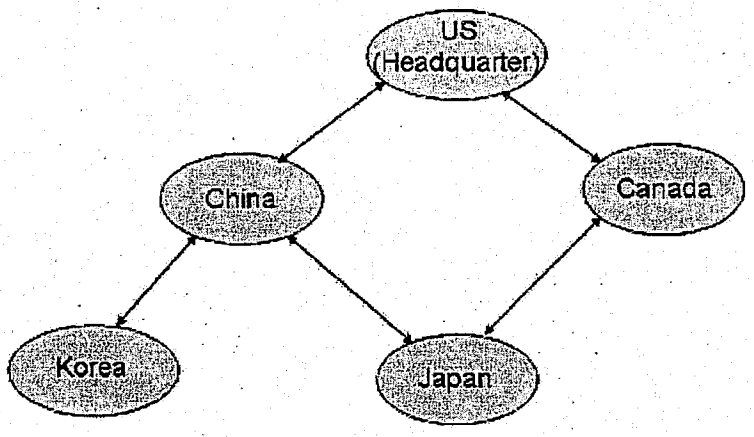


Figure 1.2 Topology of the sample peer-to-peer data warehouses

Suppose that the Headquarter requires the following decision analysis information:

- The total sales amount of each brand of product each month, quarter, and year.
- The total sales amount of each category of product each month, quarter, and year.
- Comparison of the sales performances of each country for each month, quarter, and year.

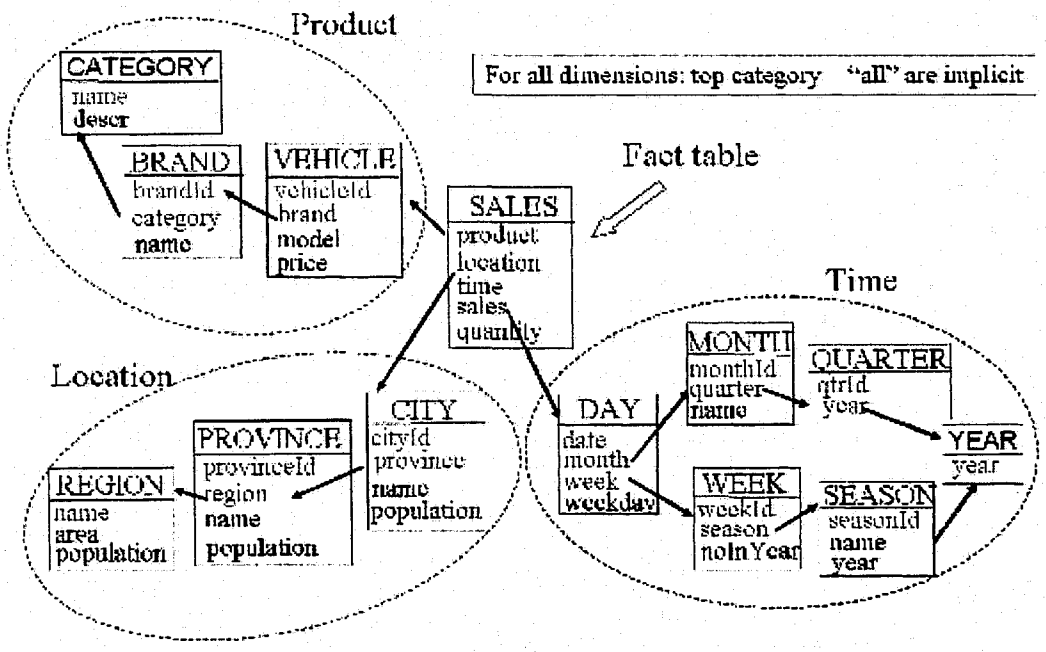


Figure 1.3 A sample data warehouse schema

To obtain this information, users at the Headquarter post the queries locally and collect the information globally. As for how to evaluate the queries globally, Espil, et al have done a large amount of work as described in Section1.1 and provided an answer in [Espil2004]. Our work focuses on investigating how to improve the system response time by selecting and materializing redundant aggregate views in different peers. In the following, we use this running example to illustrate how materialized aggregate views benefit the system performance of a peer data warehousing system. To make the problem easy to explain, we present the sample data warehouse system mentioned above as Figure 1.4; we partition the memory of each peer data warehouse into two parts: one, called local memory, stores the *local materialized views* (LMVs), the local *base fact table*, and *dimensions,* and the other, called global memory, stores *global materialized views* (GMVs). We provide the formal definition of LMVs and GMVs in Chapter 3. In addition, for each peer data warehouse, we group the queries into two sets: one contains a series of queries being answered from local

data, called *local queries* (LQs), and the other is comprised of *global queries* (GQs), which are answered from global data.

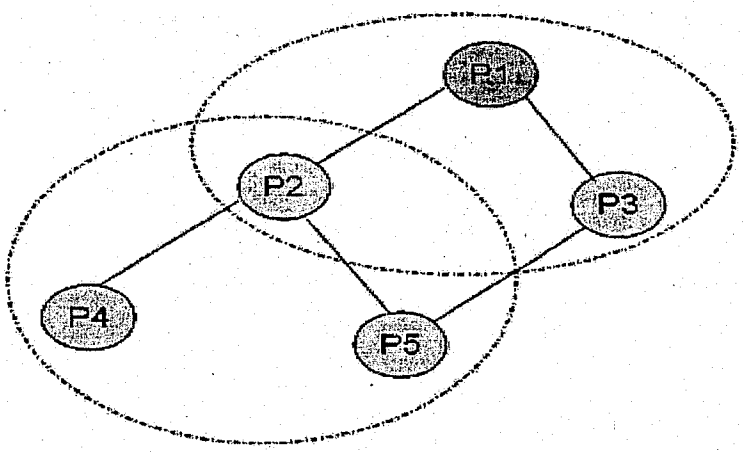


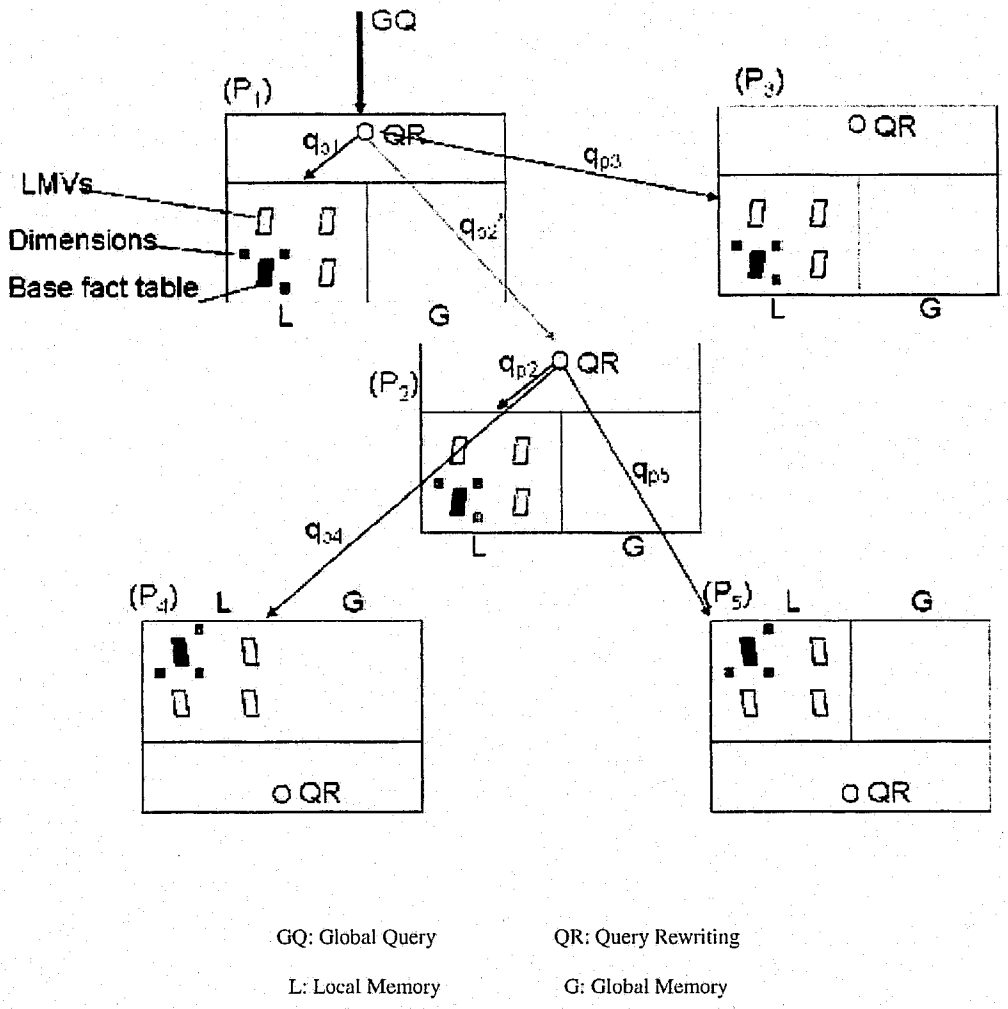
Figure 1.4 A P2P data warehousing system for the running sample

In the following, we discuss how global materialized views speed up the global queries posted on P_1 by comparing two scenarios.

Scenario 1: No GMVs stored at any locations in the system

Before GMVs are selected, no any intermediate results of sub-queries are stored as GMVs on P_1 or any other locations in the system. To answer the GQ posted on P_1, after query rewriting which we will discuss in Section 2.4, P_1 sends the query expressions to its acquaintances (P_2 and P_3) to collect the required data. After each acquaintance receives a sub-query, it will rewrite the received query and send it to its acquaintances if it has relevant peers; otherwise, it sends the result to its parent peer immediately. If the acquaintance, such as P_2, has acquaintances, it starts to compute the query expressions and transform the data when all relevant results from its acquaintances are received, and then sends the result back to its parent peer P_1. Once P_1 has received all collected data, it computes the final integrated answer. In this scenario, the computing cost should include all sub-queries computing cost

on each peer and the costs of data transfer and transformation between P_1 and P_2, P_3 as well as P_2 and P_4, P_5. This scenario is shown in Figure 1.5.



GQ: Global Query QR: Query Rewriting

L: Local Memory G: Global Memory

Figure 1.5 Query processing model in a P2P data warehousing system without GMVs

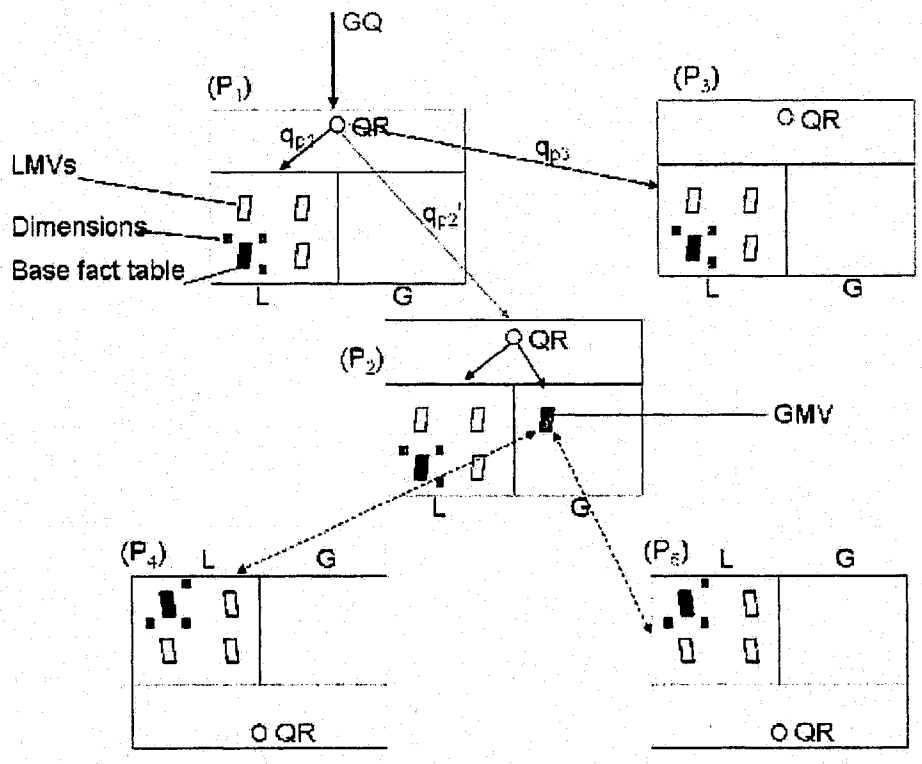
Scenario 2: Some intermediate results stored on P_1 and/or P_2 as GMVs

Case1: An intermediate query result from P_4 and P_5 for computing $q^{p_2'}$ is stored as a GMV on P_2, as shown in Figure 1.6 (case 1).

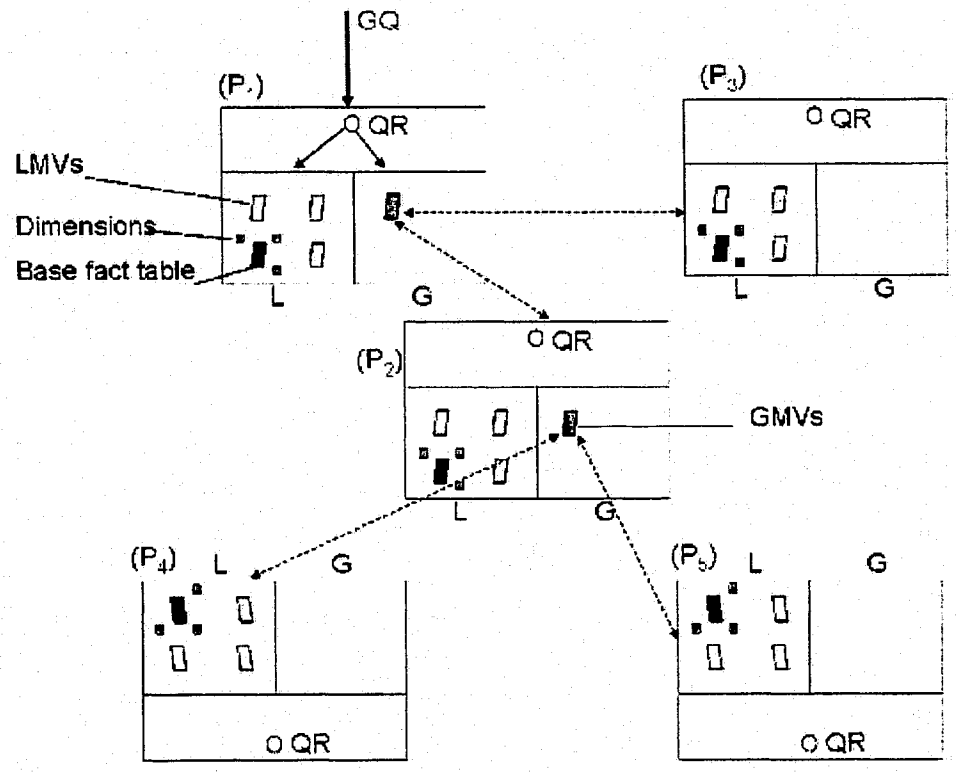
In this case, to answer the GQ posted on P_1, the rewritten queries do not need to propagate to P_4 and P_5. When the query message arrives at P_2, the system will immediately get the required data for P_4 and P_5 since it has been stored as a GMV in P_2.

Case 2: An intermediate query result from P_2, P_3, P_4, and P_5 for GQ is stored as a GMV on P_1, as shown in Figure 1.6 (case 2).

To answer the GQ posted on P_1, the system does not need to collect data from any acquaintances since the system can directly get the required data in P_1 that is stored as a GMV there. As a result, the system saves the query messaging and data transfer costs between P_1 and any other peers in the system, as well as query computing and data transformation costs on each acquaintance. However, in this case, to keep the data of the GMV in P_1 consistent with its data source distributed in other peers, we need to account for the global maintenance cost.



Case 1



Case 2

GQ: Global Query QR: Query Rewriting L: Local Memory G: Global Memory

Figure 1.6 Query processing model in P2P data warehousing system with GMVs

After proper GMVs are selected and stored in appropriate locations, the system will get the answer to the GQ more efficiently since there are no or less data transfer and transformation costs involved. As a result, the system response time is greatly improved and also the data are available even when some peers are temporarily not connected to the system. However, we have to consider the data freshness and update consistency in this case. Therefore, the global maintenance cost should be taken into account when a view is materialized. By

storing GMVs, how much benefit we can get is the difference between total query evaluation cost saving and global data maintenance cost incurred.

1.4 Contributions and Thesis Outline

This section addresses the main contributions of this thesis and outlines the rest of thesis.

Our work focuses on how to select sets of materialized aggregate views on different peers to speed up the global queries posted on a given peer in a peer data warehousing system, and meanwhile minimize the maintenance costs of selected materialized views. In this thesis, we make the following contributions:

- First, we develop a theoretical framework for the problem of selecting *global materialized views* (GMVs) in a peer data warehousing environment.

- Second, we extend the concepts of *Expression AND-DAG* [Gupta1997, Gupta1999], *query aggregation lattice* [Harinarayan1996], and *cost model* definition [Gupta1997, Gupta1999] defined on a centralized data warehouse to peer data warehouse systems. In centralized data warehouses, the *Expression AND-DAG* and *query aggregate lattice* are constructed statically and all query knowledge is regarded as given. In our problem scope, *Expression AND-DAG* and *query aggregation lattice* in a peer data warehouse system are constructed dynamically.

- Third, we propose a greedy base algorithm to solve our global materialized view selection problem. The view selection algorithm is greedy both wrt the whole system and wrt each peer. We also provide a solution to an open problem mentioned in [Espil2004], namely, the problem of storing and maintaining the redundant views with aggregation in peer data warehouses.

- Finally, we implement and simulate the P2P GMV selection algorithm with an n-dimension data warehouse system. From the experiment results, we find that the query frequencies and the update frequencies of materialized views, as well as the data transfer rate in a system, have critical impacts on the final view selection results. The longest path of a P2P network, the number of granularity levels in the global aggregate lattice, and the number of data warehouse dimensions affect the P2P GMV selection algorithm processing time, the average minimum query computing cost of P2P global queries, and the materialized view maintenance cost.

The remainder of the thesis is organized as follows. In Chapter 2, we review prior work and introduce fundamental concepts that our research depends on. In Chapter 3, we first give a formal definition of the problem, and then formally describe the problem formulation, including extended query aggregation lattice, extended Expression AND-DAG, and cost model used within the peer data warehousing scenario. After that, we present the P2P view selection greedy algorithm and illustrate the algorithm using a walkthrough example in Chapter 4. Chapter 5 first addresses the algorithm implementation setting, and then analyzes the factors that affect the final view selection results according to the experiment results. Finally, we conclude our thesis and point out potential future work in Chapter 6.

Chapter 2 Background

This chapter first discusses the related work on the view selection problem in both centralized and distributed data warehouse systems. Second, we introduce the fundamental concepts of the multidimensional data model. Following that, we review the concepts of the P2P multidimensional data model for peer data warehousing system. Finally, we define the aggregate queries and a query rewriting approach in Section 2.4.

2.1 Related work

Selecting an appropriate set of views to materialize in a data warehouse has attracted a lot of interest in recent years. Most of research work has been done with regard to the materialized view selection problem in a *centralized data warehouse*. We are also aware that there is some work discussing the view selection problem in the context of *distributed data warehouse systems*. In the following, we discuss the related work on the view selection problem in the centralized and distributed data warehouses, respectively.

First, we review related work on the centralized data warehouses. We classify the solutions to the view selection problem in centralized data warehouse systems into two categories. The first category is made of *static view selection algorithms*. These algorithms return a set of materialized view candidates for given user queries and constraints, e.g., those with a limited amount of materialization time, storage space, and view maintenance cost. The most important work done in this category is [Gupta1997].

In [Gupta1997], Gupta develops a theoretical framework for the general view selection problem in a centralized data warehouse and proposes competitive polynomial-time heuristics for selections of views to minimize the total query response time for different cases of data warehouse scenarios. He uses AND view graphs as well as OR view graphs to this end. In an AND view graph, each query/view has a unique evaluation plan; whereas in an OR view graph, a view can be computed from one of the related views. AND view graph

together with OR view graph are generalized in the notion of AND-OR view graphs. He also considers AND view graphs and OR view graphs with indexes. Gupta provides solutions to the view selection problem under the space constraint, and the constraint representing the maximum amount of disk space that can be used to store the selected materialization views.

Harinarayan et al. present a framework and a greedy algorithm to select a good set of aggregate queries to materialize for the case of data cubes based on the maximum benefit per unit space (BPUS) [Harinarayan1996]. In this case the view graph is an OR view graph. This is a special case of view selection problem mentioned in [Gupta1997]. For this special case, Harinarayan et al. propose an aggregate lattice framework to express dependencies among views, so that the view dependencies are taken into account in the process of cube view selection. Later we extend this aggregation lattice framework to solve the view selection problem in a peer data warehouse system. The disadvantage of this work is that it ignores update costs of materialized views and limits the lattice framework to the special case mentioned above.

Gupta and Mumick develop an algorithm to select a set of views to materialize in a centralized data warehouse in order to optimize the query response time under the constraint of a given total view maintenance cost [Gupta1999]. This maintenance–cost view selection problem is much harder to solve than the view selection problem under the constraint of space since the relative benefit function has non-monotonic properties. They first design an approximation greedy algorithm for the maintenance-cost view selection problem in an OR view graph, and then design an A* heuristic delivering an optimal solution for the general case of AND-OR view graphs.

Those approaches mentioned above provide the fundamental theory and techniques for solving the view selection problem in centralized data warehouse environments. Their theories and ideas are not only applied to a centralized data warehouse, but also can be extended to distributed data warehousing systems. Our work is closely related to these approaches. We extended their concepts in the context of centralized data warehouse

systems to peer data warehouse systems. How our work extends their work is discussed in details in subsequent sections.

Several further approaches to the view selection problem exist [Baralis1997, Shukla1998, Valluri2003, Gou2006]. These approaches provide similar solutions to the view selection problem in centralized data warehouse systems, but offer some improvements on various aspects of the problem. Shukla et al. revisit the BPUS algorithm proposed in [Harinarayan1996] and propose a simple and faster algorithm for some conditions [Shukla1998]. Valluri et al. design a *View Relevance Driven Selection* algorithm that minimizes the total cost of query processing and view maintenance [Valluri2003]. In their solution, only relevant views are considered when selecting views. Baralis et al. provide a solution that makes use of a reduced amount of space by reducing the number of views to consider in the aggregation lattice [Baralis1997]. Gou et al. revisit the problem of materialized view selection under disk space constraints. They provide a more powerful, efficient, and flexible approach for solving the problem by giving a new algorithm [Gou2006].

A special sample work of view selection algorithms in the *static view selection* category worth mentioning here is reported in [Sellies1998]. This work solves the view selection problem by operating on multiple view processing plans and applying multiple query optimizations. The main idea of this work is an approach for building a good overall query plan for a given query scenario.

The second main category of view selection problem in centralized data warehouse systems is *dynamic view selection algorithms*. These algorithms use the idea of caching the results of user queries to speed up similar queries of other users [Deshpande1998, Kotidis1999].

Now let us go over the related work on the view selection problem in the distributed data warehouse scenario. Yang, et al. [Yang1997] develop a framework for selecting materialized views in a distributed data warehouse environment, which is based on the specification of *Multiple View Processing Plans* (MVPPs). The goal of their work is to

achieve the best combination of good query performance and low view maintenance. The approach they use is deriving common intermediate results which can be shared by multiple queries. They also design a cost model that takes into consideration query access frequencies and base relation updating frequencies as well as query access costs and view maintenance costs. As for the view maintenance, they assume that view re-computing be used whenever the base relations updating occur. To solve the problem, they give a heuristic algorithm which can provide a feasible solution based on individual optimal query plans; after that they propose a guaranteed optimal solution by mapping this problem as 0-1 integer programming problem.

The other important paper that discusses the distributed view selection problem is [Bauer2003]. In [Bauer2003], Bauer et al. focus on solving the view selection problem in the context of distributed data warehouse architectures. The objective of their work is to select the optimal aggregate view combination for materialization, constrained by storage capacity of each node in the distributed system. They extend the concept of an aggregation lattice mentioned in [Harinarayan1996] to capture the distributed data warehouse semantics. Moreover, they extend a greedy-based view selection algorithm based on an adequate cost model for the distributed system. This paper provides an important hint for us to extend the concepts of an AND view graph, an aggregation lattice, and a cost model for centralized data warehouse systems to peer data warehouse systems. A shortcoming of the work in [Bauer2003] is that the *query aggregate lattice* between different nodes in the distributed system is constructed statically and all query knowledge is regarded as given.

Through the overview of all related work mentioned above, we believe that all these valuable work provide us with a solid knowledge and understanding of the view selection problem. To some degree, we can say that our work is an extension of some previous work in different aspects. We extend the Expression AND-DAG, cost model definition, and greedy algorithm developed by Gupta [Gupta1997, Gupta1999], the lattice framework proposed by Harinarayan et al. [Harinarayan1996], as well as the distributed lattice framework developed by Bauer [Bauer2003]. However, to the best of our knowledge, we are

the first to address the problem of selecting views to materialize in a peer data warehouse system to speed up the global queries.

2.2 Multidimensional Data Model

Data warehouse and OLAP systems support data analysis through a *multidimensional data model* [Kimbal1998, Martyn2004, Inmon2001]. A *multidimensional data model* categorizes data as facts associated with numerical measures and textual dimensions to characterize the facts. *Dimensions* are used for selecting and aggregating data at the desired level of detail. A dimension is organized into a containment-like hierarchy composed of numerous levels, each representing a level of detail required by the desired analyses. Each instance of the dimension, or dimension member, belongs to a particular level [Espil2004, Cohen2005].

Each multidimensional data model contains a finite set of dimensions to describe the fact tables. Each dimension should have an underlying *Hierarchy Schema* (HS), which is composed of different levels of categories and each category has a non-empty finite set of members. *Fact tables* are modeled based on a list of categories called *granularity*. The *granularity* refers to the level of detail of the facts stored in a data warehouse. Let **D** represent a set of dimension names for a multidimensional data model; Let **C** represent a set of categories for each dimension d; Let **MemberSe**t represent a set of members for each category c. Example 2.1 further illustrates the concepts of dimensions, categories, and members of a category.

Example 2.1: Consider one of local data warehouses of General Motor in our running example, namely the local data warehouse in Canada. The data warehouse has the following dimensions: Product, Location, and Time. Namely, **D** = {Product, Location, Time}. For each dimension, it contains different levels of categories. For the dimension Product, **C** = {VEHICLE, BRAND, CATEGORY, ALL}; for the dimension Location, **C** = {CITY, PROVINCE, REGION, ALL}; for the dimension Time, **C** = {DAY, WEEK, MONTH, SEASON, QUARTER, YEAR, ALL}. As for the members of a category, let us consider the category of province in the dimension Location, $\mathbf{MemberSet}_{\mathbf{province}}$ = {Ontario, Alberta,

Quebec, …, Manitoba}. This multidimensional model refers to Figure 1.3.

Definition 2.1 (Hierarchy Schema): A hierarchy schema (HS) is a directed graph (DG) $H = (C, \uparrow)$, where the elements of C (the nodes of DG) are categories, $\uparrow$ is a binary relation on categories. For each HS, there is a unique top category "ALL" and one or several bottom categories. The bottom categories are also called base categories. We use $\uparrow^*$ denoting the transitive and reflective closure of $\uparrow$. For every category c it holds: $c \uparrow^* ALL$ [Bertossi2005].

Example 2.2: C = {VEHICLE, BRAND, CATEGORY, ALL}, VEHICLE and BRAND $\in$ C, VEHICLE $\uparrow$ BRAND means that VEHICLE is directly connected to BRAND in H; VEHICLE $\uparrow^*$ CATEGORY means that the category VEHICLE is directly or indirectly connected to CATEGORY in H.

Definition 2.2 (Dimension Schema): A dimension schema should have an underlying hierarchy schema $H = (C, \uparrow)$ which consists of a finite set of categories, ALL is the implied top category, where the elements of C are categories, $\uparrow$ is a binary relation on categories [Bertossi2005].

Example 2.3: Let us consider the dimension Time. C = {DAY, MONTH, SEASON, YEAR, ALL}, and the hierarchy structure: DAY → MONTH → QUARTER → YEAR →ALL. DAY is the bottom category in the hierarchy structure.

Definition 2.4 (Roll-up Relation): The roll-up relation is a partial order relation ≤ between members induced by the child/parent relation. $x_1 \leq x_2$ iff $x_1 << x_2$ or $x_1 = x_2$. The roll-up mapping from a category c_1 to a category c_2 of a dimension d, denoted $R_{c1}^{c2}(d)$, is the set:

$\{(x_1, x_2) | x_1 \in MemberSet_{c1}, x_2 \in MemberSet_{c2}, x_1 \leq x_2\}$ [Bertossi2005]

Example 2.4: Let us consider a category CITY and a category PROVINCE in the dimension Location of the data warehouse located in Canada, $MemberSet_{city}$={Toronto, Ottawa, Montreal}, and $MemberSet_{province}$={Ontario, Quebec}. The Roll-up Relation between CITY and PROVINCE is:

$$R_{CITY}^{PROVINCE}(LOCATION) = \{(x_1, x_2) | x_1 \in MemberSet_{city}, x_2 \in MemberSet_{province}, x_1 \leq x_2\}$$

$$= \{(Toronto, Ontario), (Ottawa, Ontario), (Montreal, Quebec)\}.$$

Definition 2.5 (Dimension Instance): A dimension instance is a tuple d = (H, MemberSet, <), where:

- $H = (C, \uparrow)$ is a hierarchy schema
- MemberSet contains a set of members, $MemberSet_c$ for each $c \in C$
- < is a child/parent relation between members of the categories in the dimension, and MemberSet (d) := $\cup_{c \in C} MemberSet_c$.

The definition of dimension is subject to several constraints. Every dimension instance d = (H, memberSet, <) must satisfy the following conditions [Bertossi2005]:

- Disjointness: The member sets are pairwise disjoint. This means that, if categories $c_1 \neq c_2$, then $MemberSet_{c1} \cap MemberSet_{c2} = \emptyset$. This condition avoids redundant aggregates in the data-cube.
- Top category constraint: $MemberSet_{All}$={all}
- Connectivity: For two members to be directly connected via <, then it is necessary that their corresponding categories must be directly connected via $\uparrow$.
- Partitioning: For every pair of categories c, c', if there exists a member x $\in MemberSet_c$, and a pair of members $x_1, x_2 \in MemberSet_{c'}$ such that $x \leq x_1$ and $x \leq x_2$, then $x_1 = x_2$. This shows that we cannot roll-up from a member following up paths leading to two different elements in the same category.

Definition 2.5 (Homogeneous Dimension and Heterogeneous Dimension): There are two types of rollup functions: partial rollup function and total rollup function. If there is an element x, such that $x \in \text{MemberSet}_c$, without an image in $\text{MemberSet}_{c'}$ according to $R_c^{c'}$, this rollup function $R_c^{c'}$ is partial. If $c_1 \uparrow^* c_2$, then the rollup mapping $R_{c1}^{c2}(d)$ is a total function from MemberSet_{c1} to MemberSet_{c2}. When the rolling up function is total, we say the dimension is *homogeneous*, otherwise, the dimension is *heterogeneous* [Bertossi2005]. The condition of homogeneity includes the following:

- Each member in any category has no more than one ancestor in a category above it.
- Every member in a category c must have parents in all the categories above $c \in H$.

Definition 2.6 (Fact Table Schema): A fact table schema FS is a tuple (N, G, m), where N is a name, G is a granularity, and m is a measure. For example, if $G = \{c_1, c_2, .., c_n\}$ (c_n is a category), then $(x_1, x_2, \ldots, x_n) \in c_1 \times c_2 \times \ldots \times c_n$ is a point of FS. **An Instance of a fact table schema FS** is a partial function FI that maps points of FS into elements of dom(m). A base fact table instance FI_b is a fact table instance whose granularity is a base granularity, i.e. its categories are all base categories [Bertossi2005].

Example 2.5: In our sample data warehouse schema shown in Figure 1.3, the name of the fact table is SALES, m is the sales quantity, namely the number of vehicles sold. The base granularity G_b = (vehicle, city, day}.

Definition 2.7 (Cube View): A cube view over a set of dimensions d wrt a base fact table instance FI_b, a granularity G, and an aggregation function *f(m)* with measure *m*, (denoted by $CubeView_{G,f(m)}(d, FI_b)$), is defined by the following relational algebra expression [Bertossi2005]:

$$\Pi_{G,f(m)}(FI_b \bowtie R_{Gb[1]}^{G[1]}(d[1]) \bowtie \ldots \bowtie R_{Gb[n]}^{G[n]}(d[n]))$$

Where

- d is a set of dimensions: d[1], d[2], …, d[n].
- G_b is a base granularity for d consisting of the categories $G_b[1]$, … , $G_b[n]$ over d[1], d[2], …, d[n].
- FI_b is a base fact table instance wrt d with base granularity G_b and measure m.
- G is a granularity over d: G[1], … , G[n].
- $R_{Gb[i]}^{G[i]}(d[i])$ is a roll-up function from category $G_b[i]$ to category G[i] in the dimension d[i].
- f is an aggregation function on dom(m).
- $\bowtie$ is a join operator, representing the join relationship between two expressions.
- $\Pi_{G, f(m)}$ means that the final join result is projected over the attributes in G and the aggregation function f grouping by the attributes in G is applied to the result.

So by using the cube view expression, we can aggregate the base fact instance to any selected granularity by computing a usual join between FI_b and those rollup functions [Bertossi2005]. We also call a cube view an aggregate view in our data warehouse context and we use these two terms interchangeably throughout the thesis.

Example 2.6: In our running example, let us consider a cube view over a set of dimensions d = {PRODUCT, LOCATION, TIME} wrt a base fact table instance SALES, a granularity G = {BRAND, PROVINCE, MONTH}, and an aggregation function *sum(sales quantity)*, $CubeView_{G,sum(salesquantity)}(d, SALES)$ is:

$$\Pi_{G,sum(salequantity)}(\text{SALES} \bowtie R_{VEHICLE}^{BRAND}(\text{PRODUCT}) \bowtie R_{CITY}^{PROVINCE}(\text{LOCATION}) \bowtie R_{DAY}^{MONTH}(\text{TIME}))$$

Definition 2.8 (Data Cube): A data cube is a set of data cube views defined over the same dimension list, same base fact instance, and same aggregation functions [Bertossi2005].

Definition 2.9 (Summarization): A summarization from a fact table instance FI with granularity c' and measure m to a category c in the context of a single dimension d, with distributive aggregation function f, (denoted by $SUMM_{c,f(m)}(d, FI)$), is defined as the following relational algebra expression:

$$\Pi_{d, f(m)} (\text{FI} \bowtie R_{c'}^{c} (\text{d})$$

Where

- d is a dimension
- FI is a fact table instance wrt d with granularity c' and measure m.
- $R_{c'}^{c}$ (d) is a roll-up function from category c' to category c in the dimension d.
- f is an aggregation function on dom(m).
- $\bowtie$ is a join operator, representing the join relationship between two expressions.
- $\Pi_{d, f(m)}$ means that the final join result is projected on the attribute in d and the aggregation function f grouping by the attribute in d is applied to the result.

We say that a category c is summarizable from a category c' in a dimension instance d if for every distributive aggregate function f and every base fact table instance FI_b of d, it holds:

$$CubeView_{c,f(m)}(d, FI_b) = SUMM_{c,f^c(m)}(d, CubeView_{c',f(m)}(d, FI_b))$$

The LHS represents the aggregation via f from the base category up to category c, while the RHS represents first the aggregation via f from the base category up to category c' and then the aggregation via f^c, the intermediate facts from c' up to category c [Bertossi2005]. Hence, the main difference between *CubeView* and *SUMM* is that the former starts from the base category and the later from any category c' below c.

2.3 P2P multidimensional data model

The *P2P multidimensional data model* is used for designing a peer data warehouse. Following the paper [Espil2004], we summarize the concept of a peer data warehouse. A

peer data warehouse system consists of a collection of autonomous local data warehouses distributed in a P2P network. Each local data warehouse has its own schema and metadata formats. Each peer data warehouse holds only the dimension of interest at that peer, and the local fact table is constructed from the local dimensions. The base fact table for each peer data warehouse is constructed from the bottom levels of hierarchy dimensions. The peer data warehouse system allows cooperative interchange of decision-making information without a global schema between peers. Each node can publish its schema to the involved peer community so that other peers can automatically discover the specific schema. The queries are posed locally and evaluated globally. The following is the characteristics of a peer data warehouse system:

- Any peer may join or leave the P2P sharing system and contribute data, schemas, mappings, or computation to the whole system.
- Each peer has independent but semantically related schemas.
- Each peer acts as both a data origin and mediator.
- Each peer maintains full autonomy over its own data.
- No global schema is assumed to exist for shared data.
- Peers must cooperate in maintaining their respective local views.

In the following, we review the P2P multidimensional model based on the paper [Espil2004].

Definition 2.10 (Peer Dimensions): A P2P data warehouse system contains a set of dimensions $d_1, \ldots, d_r$ representing the same semantic concept d and are distributed over different peers; the set of these dimensions are called peer dimensions, denoted as P_d. Each dimension $d_i \in P_d$ is called a peer dimension wrt any other dimension $d_j \in P_d$. Each peer has at most one peer dimension.

Let N denote a set of peers in a P2P network. Given a dimension $d_i \in P_d$, and $p \in N$, to locate its peer dimension d_j at the peer p, a relation R such that each tuple (d, p) in R must be stored at every peer. The relation R(d, p) means that a dimension d is assigned to peer p. The

schema between peer dimensions may be different, so the correspondence among levels between peer dimensions and mapping among members for each pair of corresponding levels must be defined.

Definition 2.11 (Level Correspondence): Suppose (d_i, d_j) is a pair of peer dimensions which are both members of P_d, with schemas $d_i = (C_i, \uparrow)$ and $d_j = (C_j, \uparrow)$ respectively. A level correspondence from d_i to d_j (denoted by $Levels_{di}^{dj}$) is an injective partial function that maps levels in C_i to levels in C_j. An order preserving level correspondence between d_i and d_j satisfies the following conditions for each pair of levels c_1 and c_2 in dimension d_i:

- $c_1 \leq c_2$.
- If c_1 and c_2 are members of dom($Levels_{di}^{dj}$), then $Levels_{di}^{dj}(c_1) \leq Levels_{di}^{dj}(c_2)$ holds.
- $Levels_{di}^{dj}(All) = All$.

Note:

1) The above definition does not imply a complete correspondence. For example, some levels in the acquaintance dimension may not have corresponding levels in the local dimension.
2) The correspondence given by $Levels_{di}^{dj}$ may be different from that given by $Levels_{dj}^{di}$. The first one is used when d_j is the local dimension, while the second one is used when d_i is the local dimension.

A partial mapping function assigns members in the category c_{local} of the local dimension d_{local} to members in the category c_{acq} of the acquaintance dimension d_{acq}. Such a function is denoted by $map_{c_{acq} \to c_{local}}^{d_{acq} \to d_{local}}$. We call this function a *base mapping table*. If the function $map_{c_{acq} \to c_{local}}^{d_{acq} \to d_{local}}$ is total, we say that the obtained *base mapping table* is complete.

The defined correspondence functions $Levels_{d_{acq}}^{d_{local}}$ and mapping tables must be stored at each

related node, where d_{acq} stands for the acquaintance dimension, while d_{local} stands for the local dimension.

Definition 2.12 (Consistent Base Mapping Table): Let a function $Levels_{d_{acq}}^{d_{local}}$ be an order preserving level correspondence. A base mapping table $map_{c_{acq}\to c_{local}}^{d_{acq}\to d_{local}}$ is consistent if and only if the following holds:

For each member m of a category such that $map_{c_{acq}\to c_{local}}^{d_{acq}\to d_{local}}$ (m) is defined, if there exists some member m' satisfying that $Rup_{c'_{acq}\to c_{acq}}^{d_{acq}}(m')$, for some category $c'_{acq} \in$ dom($Levels_{d_{acq}}^{d_{local}}$), then

$$rup_{c'_{local}\to c_{local}}^{d_{local}}(map_{c'_{acq}\to c'_{local}}^{d_{acq}\to d_{local}}(m')) = map_{c_{acq}\to c_{local}}^{d_{acq}\to d_{local}}(m)$$

where $c'_{local} = Levels_{d_{acq}}^{d_{local}}(c'_{acq})$

Definition 2.13 (Peer Fact Tables): There exists a generic fact table f that has a schema f = (G, m), where G stands for granularity, and m stands for measures. In the P2P multidimensional model, there exists a set $\{f_1,\ldots, f_r\}$ of fact tables called peer fact tables, which have the same measures and correspond to the same generic fact table f. All dimensions in f must be included in the set of dimensions in the granularity for each peer fact table that specifies f [Espil2004].

Similar to peer dimensions, there is a relation R such that each tuple (f, p) in the relation means that a fact table f is assigned to peer p. Let N denote a set of peers in a P2P network. Given a peer fact table f_i, there is a partial function FtoG with signature F $\rightarrow$ G that maps a peer fact table f_i to its generic fact table f. Given a generic fact table f, and a node p$\in$ N, there is a partial function GtoF with signature G $\times$ N $\rightarrow$F that returns the peer fact table f_i in peer p that is generated by generic fact table f. Both FtoG and GtoF must be stored in the related peers.

2.4 Aggregate Queries and Query Rewriting in Peer Data Warehouses

Aggregate queries are extensively used in data warehouse and decision support application systems. The queries containing aggregate functions are called aggregate queries. To summarize the data in a fact table, aggregate queries involving an aggregate function such as SUM(), AVG(), MAX(), MIN(), COUNT() are often created [AFF].

In a peer data warehouse, an aggregate query q over a multidimensional space (d_1, …, d_n), posed on some peer n_l can be expressed as follows [Espil2004]:

$$q(Z_1,...,Z_n,aggr(M),N) \leftarrow Fact(X_1,...,X_n,M)$$

$$rup^{d_1}_{cbottom_1 \to c_{t_1}}(X_1,Z_1)$$

$$\cdots$$

$$rup^{d_n}_{cbottom_n \to c_{t_n}}(X_n,Z_n)$$

Where,

- d_i, i = 1, …, n are different dimensions in peer n_l with schema d_i = (C_i, ↑)
- cbottom$_i$ is the category at the bottom level of d_i, and cbottom$_i$ ∈ C_i
- c_{t_i} is a some category of d_i, and c_{t_i} ∈ C_i
- Fact is a generic fact table such that there exists a fact table baseFact = GtoF(Fact, n_l) with schema Fact = (G, M), and G = (d_1,...,d_n)
- aggr is a distributive aggregate function, compatible with measure M
- $rup^{d_i}_{cbottom_i \to c_{t_i}}$, i =1, …, n, are roll-up functions

When a query q is posted on one peer in a peer data warehouse system, to evaluate this query globally, query rewriting must be accomplished. Before query rewriting is done, we must first identify which peers are relevant to the query so that we can decide which acquaintances to send the rewritten query q to.

Definition 2.15 (Relevant Peer): A peer p∈ N is relevant for a P2P – OLAP query q if and only if the following conditions are satisfied [Espil2004]:

- There exists a fact table baseFact$_p$ in p, such that basefact$_p$ = GtoF(Fact, p)
- For each dimension d_k, k=1, …, n, in the schema of Fact, there exists a category c_{s_k} in d_k such that a pair (c'_{s_k}, c_{s_k}) exists in $Levels_{d'_k}^{d_k}$; d'_k is the peer dimension of d_k in node p, and category c_{s_k} precedes or is equal to category c_{t_k} in query q.

After the relevant peers are discovered, the aggregate query q can be rewritten as follows [Espil2004]:

$$q(Z_1,...,Z_n, Ag(\{...,M_j,...\})) \leftarrow ...,$$
$$q^p(Z_1,...,Z_n,M_j),$$
$$...$$ (1)

where the query q^p, for any peer p relevant to query q, can be defined as:

$$q^p(Z_1,...,Z_n, Ag(M)) \leftarrow q_p(B_1,Y_1,...,B_n,Y_n,M),$$
$$rup_{B_1\rightarrow c_{t_1}}^{d_1}(Y_1,Z_1),$$
$$...,$$
$$rup_{B_n\rightarrow c_{t_n}}^{d_n}(Y_n,Z_n)$$ (2)

where B_k, k=1, …, n are level variables; Y_k and Z_k are members at B_k and C_{tk}, respectively. The query $q_p(B_1,Y_1,...,B_n,Y_n,M)$ is considered extensional since we assume the query result is known here.

Example 2.7: Suppose a global query (GQ) over dimensions (Time, Product) is posed to P_1 in Figure1.4, which can be expressed as the following datalog expression [Harinarayan1996] :

$$q(Z_1,Z_2,SUM(M),\{p_1,p_2,p_3\}) \leftarrow SALES(X_1,X_2,M),$$ (1)

$$\text{rup}^{TIME}_{DAY \to MONTH}(X_1, Z_1),$$

$$\text{rup}^{PRODUCT}_{VEHICLE \to CATEGORY}(X_2, Z_2)$$

This query asks for the total amount of vehicle sales by month and category in the concerned peer data warehousing system.

After query rewriting, the global query can be rewritten as:

$$q(Z_1, Z_2, SUM(\{M_1, M_2', M_3\})) \leftarrow q^{p_1}(Z_1, Z_2, M_1), \quad (2)$$
$$q^{p_2}(Z_1, Z_2, M_2'),$$
$$q^{p_3}(Z_1, Z_2, M_3)$$

The subquery $q(Z_1, Z_2, SUM(M_2'), \{p_2, p_4, p_5\}) \leftarrow q^{p_2}(Z_1, Z_2, M_2')$ can be further written as:

$$q(Z_1, Z_2, SUM(\{M_2, M_4, M_5\})) \leftarrow q^{p_2}(Z_1, Z_2, M_2), \quad (3)$$
$$q^{p_4}(Z_1, Z_2, M_4),$$
$$q^{p_5}(Z_1, Z_2, M_5)$$

As a result, the above queries can be computed by using the following datalog expressions:

$$q^{p_1}(Z_1, Z_2, SUM(M)) \leftarrow q_{p_1}(B_1, Y_1, B_2, Y_2, M), \quad (4)$$
$$\text{rup}^{TIME}_{B_1 \to MONTH}(Y_1, Z_1),$$
$$\text{rup}^{PRODUCT}_{B_2 \to CATEGORY}(Y_2, Z_2)$$

$$q^{p_2}(Z_1, Z_2, SUM(M)) \leftarrow q_{p_2}(B_1, Y_1, B_2, Y_2, M), \quad (5)$$
$$\text{rup}^{TIME}_{B_1 \to MONTH}(Y_1, Z_1),$$
$$\text{rup}^{PRODUCT}_{B_2 \to CATEGORY}(Y_2, Z_2)$$

$$q^{p_3}(Z_1, Z_2, SUM(M)) \leftarrow q_{p_3}(B_1, Y_1, B_2, Y_2, M), \quad (6)$$
$$\text{rup}^{TIME}_{B_1 \to MONTH}(Y_1, Z_1),$$
$$\text{rup}^{PRODUCT}_{B_2 \to CATEGORY}(Y_2, Z_2)$$

$$q^{p_4}(Z_1, Z_2, SUM(M)) \leftarrow q_{p_4}(B_1, Y_1, B_2, Y_2, M), \quad (7)$$
$$\text{rup}^{TIME}_{B_1 \to MONTH}(Y_1, Z_1),$$
$$\text{rup}^{PRODUCT}_{B_2 \to CATEGORY}(Y_2, Z_2)$$

$$q^{p_5}(Z_1, Z_2, SUM(M)) \leftarrow q_{p_5}(B_1, Y_1, B_2, Y_2, M), \quad (8)$$
$$\text{rup}^{TIME}_{B_1 \to MONTH}(Y_1, Z_1),$$
$$\text{rup}^{PRODUCT}_{B_2 \to CATEGORY}(Y_2, Z_2)$$

In the above formulas, predicates q_{p_k} in the expressions (4), (5), (6), (7), (8), k = 1, 2, 3, 4, 5 are considered extensional; while q^{p_k} are considered virtual.

2.5 Conclusion

This chapter has systematically reviewed the previous work that is closely related to our thesis work. This previous work builds the foundation of our thesis. The basic concepts of the multidimensional data model, the P2P multidimensional data model, and aggregate query introduced in this chapter is essential to understand the rest of this thesis.

Chapter 3 Formal Problem Formulation

In this chapter, we first give a formal definition of a *peer data warehousing system*, including the notions of *local query*, *global query*, *local materialized view*, and *global materialized view* defined on a peer data warehousing system. Second, we introduce the concepts of the *P2P aggregate lattice* framework and *P2P Expression AND DAG*. Third, we give a formal definition of the problem this thesis is dealing with. We also describe how the *P2P aggregate lattice* framework is constructed and we establish a cost model for the P2P view selection problem. Finally, based on the developed cost model, we define the notion of the benefit of a selected view, which our greedy algorithm relies on.

3.1 Peer Data Warehousing System

Assume a model which contains N peers, each peer holds a local data warehouse, called peer data warehouse, and each peer P_i is associated with a set of global queries $Q_i=\{ q_{i1}, q_{i2},..., q_{im} \}$, where each query q_{ik} has a non-negative weight f_{ik} standing for the query frequency. Each connected pair of P_s and P_t peers are connected by an edge e_{st} with a communication cost w_{st} per unit of data transferred. In the following we give formal definitions of a peer data warehouse system and its associated concepts.

Definition 3.1 (Peer Data Warehousing System): A peer data warehousing system Φ is constituted of a set of peer data warehouses, each of which consists of a fact peer *f*, peer dimensions d, a set of local materialized views LMVs, a set of global materialized views GMVs, and a set of mappings $\mathcal{M}$ specifying the semantic relationships with the related acquaintance peers in both the schema level and the instance level. Formally, each peer data warehouse P$\in \Phi$ is defined as a tuple P = (*f*, d, LMVs, GMVs, $\mathcal{M}$). Each pair of P_s and P_t peers is connected by an edge e_{st} with a communication cost w_{st} per unit of data transferred. An edge e_{st} between peers P_s and P_t is called an acquaintance between P_s and P_t. We will abuse the notation by calling P_t the acquaintance of P_s and vice versa. As a result, the

system can be represented as $\mathcal{P} = \{P, E\}$; P stands for a set of peers and E stands for a set of connected edges between two connecting peers. LMVs and GMVs are allowed to be empty.

Definition 3.2 (Local Query (LQ)): A local query is a query that is solely answered by the data from local materialized views or the base fact table and dimensions in a local data warehouse. In other words, only the local data is needed to answer a local query.

Definition 3.3 (Global Query (GQ)): A global query can be answered by the data from LMVs, GMVs, the base fact table and dimensions in a local data warehouse, and/or the data retrieved from its acquaintances recursively. In other words, peers must cooperate with each other to answer a global query. Not only local data but also global data (data distributed in other peers) are needed to answer a global query.

Definition 3.4 (Local Materialized View (LMV)): An LMV is a view that is constructed from the data within a local data warehouse, selected by executing the local view-selection algorithm, and maintained by synchronizing the data with the fact table and dimensions or other LMVs in the same local data warehouse.

Definition 3.5 (Global Materialized View (GMV)): A GMV is a view that is selected to speed up the process of answering global queries by running the global view-selection algorithm, and is constructed from the data from relevant peers recursively to eliminate data transfer costs between peers as well as view re-computing costs on the associated acquaintances. GMVs are maintained by synchronizing the data with the remote source views from related acquaintances and/or other GMVs in the same aggregation lattice.

3.2 P2P Query Aggregation Lattice

In the following, we first review the concept of the lattice framework proposed in [Harinarayan1996] ; then we extend this concept to a P2P data warehouses system. Queries on a data warehouse or OLAP system can be modeled as aggregate queries by the data cube

operators, such as sum, max, etc. The dependency relation between aggregate views in the data cubes can be described using a lattice framework.

Definition 3.6 (Hypercube Lattice ($\mathcal{L}$)): Aggregate views (also called aggregate queries) in the lattice framework are vertices of an n-dimensional cube. The following properties define the hypercube lattice of aggregates [Harinarayan1996]:

(1) There exists a partial order ≤ between aggregate views in the lattice. For aggregate views u and v, u ≤ v if and only if u can be answered using the results of v by itself. We say that u is dependent on v.

(2) There is a base view in the lattice, upon which each view is dependent. The base view is the fact table.

(3) There is a completely aggregate view "ALL" which can be computed from any other view in the lattice.

(4) ancestor(u) = {v | u ≤ v}

descendant(u) ={v |v ≤ u}

parent(u) = { v | u < v; ¬∃w, such that u < w, w < v}

children(u) = { v | v < u; ¬∃w, such that v < w, w < u}

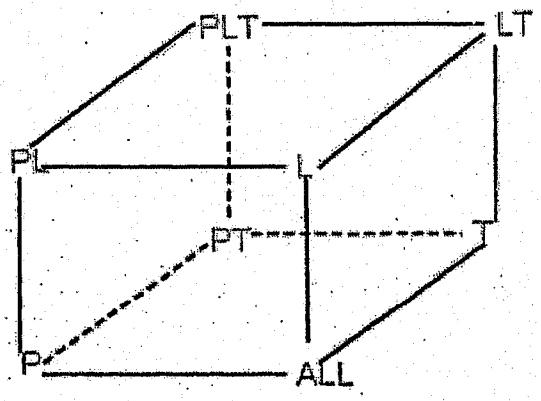


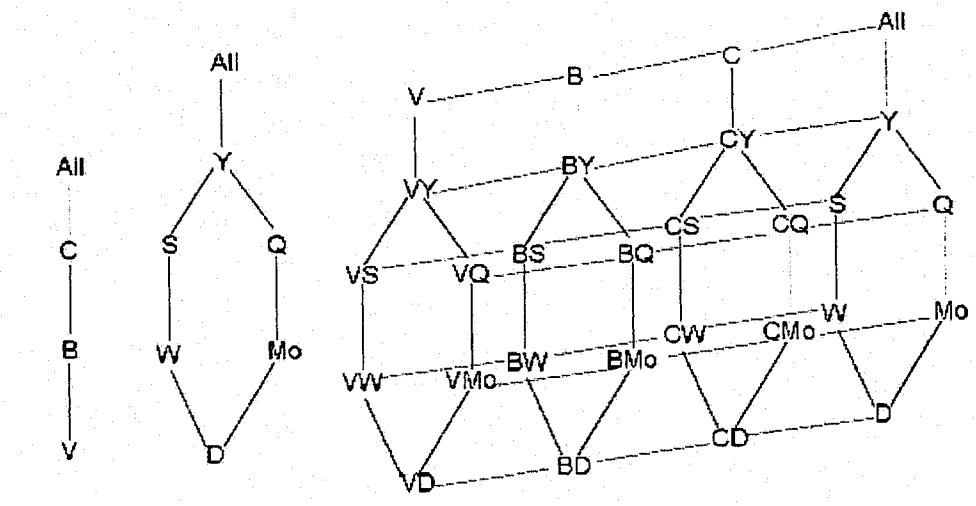
Figure 3.1 Hypercube lattice without dimension hierarchy

Example 3.1: Let's use the sample schema shown in Figure 1.3 to illustrate the concept of hypercube lattice. In this sample data warehouse schema, there are three dimensions: Product P, Location L, and Time T. Here we ignore the hierarchy of each dimension. The hypercube lattice for the sample schema is shown in Figure 3.1. The lattice includes all anticipated aggregate queries constructed from the given dimensions.

Definition 3.7 (Composite Lattice): A composite lattice is a lattice constructed with multiple and hierarchical dimensions. There are two types of query dependencies in the lattice: one is the query dependency caused by the interaction between different dimensions in the data warehouse schema; the other is the query dependency within a dimension caused by attribute hierarchies. Let a_i be the point in the hierarchy for the ith dimension, then the dependency of two points in the composite lattice can be described by the following rule [Harinarayan1996]:

$$(a_1, a_2, ..., a_n) \leq (b_1, b_2, ..., b_n) \text{ if and only if } a_i \leq b_i \text{ for all } i$$

Example 3.2: Figure 3.2 (c) shows an example of composite lattice, which is constructed from two hierarchy dimensions, (i) the Product dimension as shown in Figure 3.2 (a) where V, M, and C stand for Vehicle, Brand, and Category; (ii) the Time dimension as shown in Figure 3.2 (b), where D, W, S, Mo, Q, and Y stand for Day, Week, Season, Month, Quarter, and Year, respectively. So VD stands for VEHICLEDAY; VW stands for VEHICLEWEEK; and so on.



(a) Product (b) Time (c) Composite lattice with the combination of two hierarchical dimensions

Figure 3.2 Composite lattice with multiple and hierarchy dimensions

Definition 3.8 (P2P Generic Aggregate Lattice ($\mathcal{L}_g$)): A P2P generic aggregate lattice is constructed from the common dimensions (peer dimensions) of a P2P multidimensional model. Each view in this lattice is an aggregate view based on peer dimensions of the data model. A P2P generic aggregate lattice is a subset of aggregate lattice of any local data warehouse in the system in the context of dimensions combination. For the whole system, there is only one P2P generic aggregate lattice. Figure 3.3 shows the P2P generic aggregate lattice of our running example. The following properties define this generic lattice:

(1) There exists a partial order ≤ between aggregate views in the lattice. For aggregate views u and v, u ≤ v if and only if u can be answered using the results of v by itself. We say that u is dependent on v.

(2) There is no physical base view in the lattice, upon which each view is dependent. Each view in this lattice may be computed in two ways to compute. One way is that it can be computed from another view in the same lattice; the other way is that it can be computed from its P2P Expression AND-DAG, which we will discuss in the next section.

(3) ancestor(u) = $\{v \mid u \leq v\}$

descendant(u) ={v |v ≤ u}

parent(u) = { v | u < v; ¬ ∃w, such that u < w, w < v}

children(u) = { v | v < u; ¬ ∃w, such that v < w, w < u}

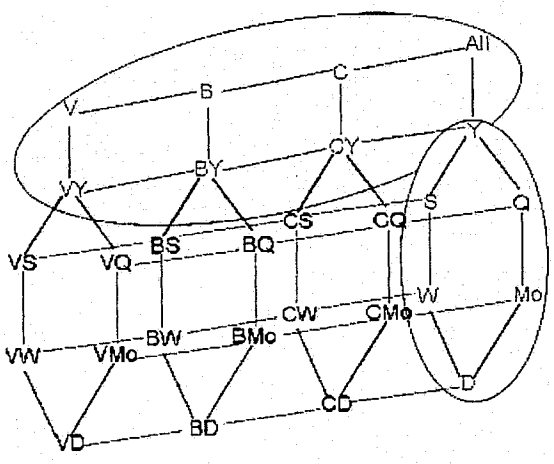


Figure 3.3 A sample of P2P generic aggregate lattice

Definition 3.9 (Global Aggregate Lattice ($\mathcal{L}_q$)): This lattice is a subset of the P2P generic aggregate lattice defined above. This means that the set of views in $\mathcal{L}_q$ is a subset of views in $\mathcal{L}_g$. It is constituted of all global queries posted on a specific peer in a P2P data warehousing system. Since the global queries posted on different peers may be different, global aggregate lattices on different peer may be different. $\mathcal{L}_q$ is a partial n-dimensional cube. It inherits the properties of the P2P generic aggregate lattice.

The difference between the local aggregate lattice $\mathcal{L}$ and the global aggregate lattice $\mathcal{L}_q$ are as follows:

(1) $\mathcal{L}$ involves all dimensions in a local data warehouse, while $\mathcal{L}_q$ only includes the common dimensions, in other words, a subset of dimension peers.

(2) $\mathcal{L}$ is a complete aggregate lattice, which includes all possible views constructed from n dimensions and all attributes on each dimension. $\mathcal{L}_q$ is a partial aggregate lattice, in other

words, the vertices of $\mathcal{L}_q$ is a subset of vertices of $\mathcal{L}$. Each view in $\mathcal{L}_q$ can be computed in two ways. One way is computed from the associated P2P Expression AND DAG. The other way is computed from its parent views in the lattice $\mathcal{L}_q$; while the view in $\mathcal{L}$ only can be computed from its parent views in $\mathcal{L}$.

For example, the diagram that is highlighted in blue color in Figure 3.3 is a typical global aggregate lattice. $\mathcal{L}_q$ at most can cover all the vertices of $\mathcal{L}_g$.

Definition 3.10 (Global Aggregate Sub-Lattice): A global aggregate sub-lattice is a lattice constructed from the rewritten sub-queries of global queries in the global aggregate lattice or its decedent aggregate lattice. A peer constructs and sends its rewritten global query sub-lattice to its acquaintances according to the peer relevance. If a query in the global lattice is relevant to an acquaintance, the associated sub-query of the global query will be involved in the rewritten global aggregate sub-lattice for that targeted acquaintance; otherwise, it will be eliminated from the rewritten sub-query lattice of the targeted acquaintance.

3.3 P2P Expression AND-DAG

This section defines the notion of P2P Expression AND-DAG and uses an example to illustrate how to build a P2P expression AND-DAG for a given query. The following definition is a substantial extension of a similar one which is given in [Gupta1997].

Definition 3.11 (P2P Expression AND-DAG): An expression AND-DAG for a view V in a P2P data warehouse is a directed acyclic graph (DAG) having the base fact table or existing materialized views (LMVs or GMVs) distributed in the local or acquaintance data warehouses as "sinks" and the view V as a "source". If a view u in the DAG has outgoing edges to views $v_1, v_2, ..., v_k$, then u should be computed from all $v_1, v_2, ..., v_k$ views. The dependences between u and $v_1, v_2, ..., v_k$ are indicated by drawing a semicircle, called an AND arc, through the edges (u, v_1), (u, v_2), ..., (u, v_k), and each edge is associated with the

cost incurred during computing u from $v_1, v_2, ..., v_k$. If an edge connecting two views and the two views are distributed in different peer data warehouses, the associated cost is equal to a data transfer cost plus a data reading cost. If the two views are in the same data warehouse, then the associated cost of an edge is just a data reading cost. The evaluation cost of u in an expression AND-DAG is the sum of the costs associated with the edges and the costs associated with its descendents' AND arcs. The computation of u requires the operation, aggregation (in our concern case). We illustrate this definition in Figure 3.4.

The characteristics of the P2P expression AND-DAG include the following:

(1) It is constructed dynamically when the rewritten global queries are propagated into the P2P network. As a result, the query cost of V has to be computed bottom up.
(2) Views in a P2P expression AND-DAG are distributed in different data warehouses, so data transfer cost (the cost occurred when data is transferred between peers) should be taken into consideration when evaluating the query computing cost. Also, if the schemas in different data warehouse are different, mappings at the schema and instance levels should be taken into account.

Figure 3.4 shows how to build a P2P Expression DAG for a given view and compute the view using the constructed P2P Expression DAG. We assume all peers are relevant, the views in blue color are materialized, and the views in pink color are not materialized. For example, to evaluate the query cost of Y posted on P_1, we need to follow the following steps:

(1) The global query Y is rewritten and propagated to P_1's acquaintances.
(2) When P_2 receive the sub-query Y_2', since P_2 is not a leaf node in the system and detects that there are other relevant nodes to this query, then Y_2' is rewritten again and the rewritten sub-queries are propagated to its neighbors except the sender.
(3) The computing cost of Y_2' is the transfer costs of (Y_4+Y_5) plus the local computing costs of Y_2, Y_4, and Y_5.

(4) After P_2 obtains the computing result of Y_2', it sends this result to P_1, now we get the query cost of Y on P_1, which is the transfer costs of (Y_2'+Y_3) plus the local/global computing costs of Y_1, Y_2', and Y_3.

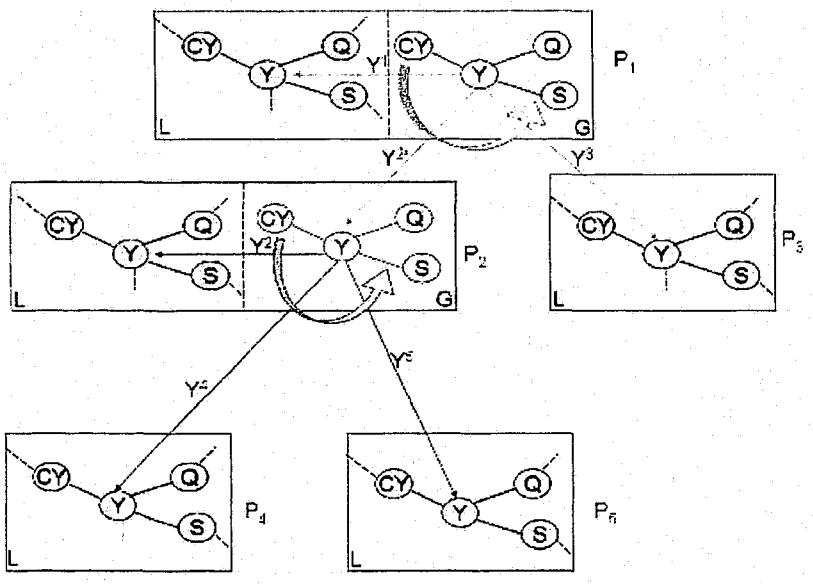


Figure 3.4 A P2P expression AND-DAG

Definition 3.12 (P2P Global Aggregate Lattice Framework ($\mathcal{L}_{p2p}$)): The P2P global aggregate lattice is constituted of global aggregate lattice on the initial peer, its sub-lattice, and its sub-sub-lattice or the sub-lattice of its sub-lattice distributed in the relevant peers, which are detected recursively in the system. In addition, the sub-lattices are connected by the edges in the P2P Expression AND-DAG graph. This whole framework is set up dynamically during the P2P global queries propagating process just like in the construction of the P2P Expression AND-DAG.

3.4 Formal Problem Definition

The problem to be solved by this thesis is to select sets of materialized aggregate views on different peers to speed up the global queries posted on a given peer in a P2P data

warehousing system such that the total cost of answering the global queries and maintaining the selected materialized views is minimized. The following is the formal definition of this problem:

Input:

- A set of peers P distributed in the P2P network G, P = { $p_1, p_2, ..., p_n$ }.
- A set of global queries posted on the peer p_l, $Q_l = \{q_1, q_2, ..., q_m\}$, each query is associated with a query frequency.
- Peer fact table $F = \{ f_1, f_2, ..., f_n \}$, f_i is the base fact table for peer p_i.
- Peer dimensions D = {d_1, d_2, …, d_s}
- A local aggregate lattice and a set of LMVs for each peer. Each query view or LMV in the lattice is associated with the following attributes: status (materialized or not), size, updating frequency, minimum computing cost, and minimum maintenance cost.

Output:

A set of GMVs on p_l and sets of intermediate GMVs on other peers to be materialized, such that the combination of the total global query cost to answer Q_l and the global maintenance cost of selected materialized views is minimized.

3.4.1 Constructing P2P aggregation lattices and P2P Expression AND-DAGs for views in the lattice

The P2P aggregation lattice can be constructed by the following steps:

(1) Construct a P2P generic lattice using the P2P multidimensional data model for the given peer data warehouse system;

(2) Construct a global aggregate lattice using the given global queries on p_l, which is used as the initial global query lattice of the P2P global query lattice framework.

(3) Construct a P2P global aggregate lattice dynamically according to node relevance of rewritten queries and P2P expression AND-DAGs for those global queries when

rewritten queries are propagated in the network. Notice that each global query is the source view for its associated P2P AND-DAG.

Recall that the view selection problem in a P2P data warehouses system can be described as: given a P2P global aggregate lattice $\mathcal{L}_{p2p}$, the view-selection problem is to select sets of materialized views $M = \{M_1, M_2, \ldots, M_k\}$ on different peers, a subset of the nodes in $\mathcal{L}_{p2p}$ that minimizes the sum of the total query cost and the total global maintenance cost of selected views.

3.4.2 Cost model

Since our algorithm requires the knowledge of the sizes of associated views, these sizes should be obtained ahead of time. We assume that we have already obtained the size of each view by using the sampling method similar to the one mentioned in [Harinarayan1996].

As mentioned before, a query in the P2P initial global aggregate lattice L_q on the peer p_i in the presence of a set of global materialized views M can be answered in two ways: (1) It can be answered from its parent views in L_q; (2) It can be answered from its P2P expression AND-DAG. Now we discuss the query evaluation cost for different cases:

Case 1: The query v can de derived directly or indirectly fro: ı its parent view in L_q
The evaluation cost of v is equal to the reading cost of its parent view. Let S_P stand for the size of the parent view. Let $QC_D(v, M)$ stand for the evaluation cost of view v in this way. To make things simple, we assume that the size of the parent view is equivalent to the cost of reading the view itself; i.e., $QC_D(v, M) = S_p$.

Case 2: The query v is answered from its P2P expression AND-DAG
The evaluation cost v includes the sum of evaluation costs of all associated views on the neighbors and the sum of the data transferring cost of all associated views from the

neighbors to where v locates. We ignore the aggregate computing cost that associates with the AND arc in v's P2P expression AND-DAG.

To make things simple, let the data transfer cost between peers be equivalent to w multiplied by the size of the data transferred, where w is the ratio of unit data transferring cost to unit data reading cost.

Formally, let QC(v, M) denote the cheapest cost of answering a query v (a node in P2P global aggregation lattice $\mathcal{L}_{p2p}$) in the presence of sets of global materialized views in the lattice. QC(v,$\varnothing$) can be calculated from v's P2P expression AND-DAG. τ (G, $\mathcal{L}_{p2p}$, M) denotes the total query evaluation and global materialized views maintenance cost in the presence of M in the P2P data warehousing system. G represents a P2P network. Each view in $\mathcal{L}_{p2p}$ is associated with a query frequency f_v and an update frequency g_v when this view is materialized.

1) Computing the query evaluation cost of a view v in $\mathcal{L}_{p2p}$

To compute QC(v, M) for a view v, we need to find the cheapest way to evaluate it by using the following steps:

First step: Let's find the cheapest cost of computing QC(v, M) in its home global aggregate lattice located on the peer where v is, denoted by $QC_D(v, M)$.

Second step: Compute QC(v, M) using its P2P expression AND-DAG. In this step, the system has to identify all associated views in its acquaintances recursively. The computing cost of QC(v, M) by using its P2P expression AND-DAG, denoted by $QC_A(v, M)$, is equal to the sum of the computing cost of its all associated views plus the data transfer cost of associated views from the sinks of its P2P expression AND-DAG to where v is, plus the aggregate computing cost associated with the AND arc in its P2P expression AND-DAG (we ignore this cost in our concern) .

Third step: Compare $QC_D(v, M)$ and $QC_A(v, M)$, and find the cheaper computing cost of $QC(v, M) = \min(QC_D(v, M), QC_A(v, M))$.

2) Total evaluation cost of global queries

Let C(G, L_q, M) denote the evaluation cost of all GQs posted on a peer p_l. To calculate C(G, L_q, M), we need to consider the query frequency of each GQ.

$$C(G, L_q, M) = \sum_{v \in L_q} f_v QC(v, M)$$

Where

- f_v is the query frequency of node v in L_q.
- L_q is the global query lattice located on p_l
- $v \notin M$ but $v \in L_q$
- M is the set of materialized views on p_l

3) Computing the global maintenance cost of a materialized view v in $\mathcal{L}_{p2p}$

Computing UC(v) for a selected global materialized view v in L_q happened at the same process of computing QC(v, M). Similarly to calculating the evaluation cost of a view, the maintenance cost of one view v in L_q may be calculated in two ways. To compute UC(v) for a materialized view v, we need to find the cheapest way to maintain it by using the following steps:

First step: Find the cheapest cost for maintaining v from its parent views in its home global aggregate lattice L_l located on the peer where v is stored, denoted by $UC_D(v)$.

Second step: Compute UC(v) using its P2P expression AND-DAG. In this step, the system has to identify all associated materialized views in its acquaintances recursively. The maintenance cost of v denoted by $UC_A(v)$, is equal to the sum of the maintenance cost of all associated materialized views plus the data transfer cost of associated views from the sinks of its P2P expression AND-DAG to p_l.

Third step: Compare $UC_D(v)$ and $UC_A(v)$, and find the cheaper maintenance cost of v, i.e, $UC(v) = \min(UC_D(v), UC_A(v))$.

4) Total GMVs maintenance cost

To calculate the total maintenance cost of materialized views on the peer p_l, we also need to consider the update frequency of each materialized view. It is worth mentioning here that the redundant view maintenance cost should be eliminated when calculating the GMVs maintenance cost globally in $\mathcal{L}_{p2p}$. This will be illustrated later in a walkthrough example.

$$U(G, L_{p2p}, M) = \sum_{v \in M} g_v UC(v)$$

Where

g_v is the updating frequency of v in the set M of materialized views of $\mathcal{L}_{p2p}$

$UC(v)$ is the maintenance cost of v

5) Total query computing cost and GMVs maintenance cost

$$\tau(G, \mathcal{L}_{p2p}, M) = C(G, L_q, M) + U(G, \mathcal{L}_{p2p}, M)$$

$$= \sum_{v \in L_q} f_v QC(v, M) + \sum_{v \in M} g_v UC(v)$$

Observation 1: $\tau(G, \mathcal{L}_{p2p}, M) = \tau(G, L_q, M_q)$

$$= C(G, L_q, M_q) + U(G, L_q, M_q)$$

$$= \sum_{v \in L_q} f_v QC(v, M_q) + \sum_{v \in M_q} g_v UC(v)$$

3.4.3 Benefit function of a selected view

Let v be an arbitrary aggregate view in a P2P global aggregate lattice $\mathcal{L}_{p2p}$. Let $B(v, M)$ denote the benefit of v with respect to M, an already selected set of views. $B(v, M) = \tau(G, \mathcal{L}_{p2p}, M) - \tau(G, \mathcal{L}_{p2p}, M \cup v)$, here τ function has been defined in the above. Since there are

view dependencies in the initial global query lattice L_k of each sub-P2P global query lattice, some view in L_k may be derived from its parent view. To make things simple, let $QC_A(v, M)$ be the evaluation cost of v computing from its P2P expression AND-DAG, $QC_D(v, M)$ be the evaluation cost of v computing from its local aggregate lattice, and $UC(v)$ be the global maintenance cost of v. Let $LRC(v)$ be the local reading cost of the view v. we redefine $B(v, M)$ as follows:

1. For each $w \leq v$ in L_k, define the benefit B_w with respect to M
   a) When $M=\emptyset$, $B_w = QC_A(w, M) - QC_D(w, M)$, $QC_D(w, M)$ is the derived computing cost of w from v.
   b) When $M \neq \emptyset$, let u be the view such that $\sum_{w \leq u} QC_D(w, M)$ is minimal and $u \in M$, $B_w = QC_D(w, M)$ from u - $QC_D(w, M)$ from v.

2. For all $w \leq v$, when it can be directly or indirectly derived from v,
   if $QC_A(v, M) - UC(v) - LRC(v) + \sum_{w \leq v} B_w > 0$,
   then define $B(v, M) = QC_A(v, M) - UC(v) - LRC(v) + \sum_{w \leq v} B_w$; otherwise $B(v, M) = 0$.

3.5 Conclusion

This chapter formally defined the peer data warehousing system and the P2P view selection problem. We extended the concept and framework of aggregate lattice and *Expression AND DAG* for a centralized data warehouse to our peer data warehousing system. Finally, we established the cost model for the P2P view selection problem and the benefit function of a selected view in a peer data warehouse system.

- For each peer data warehouse, the LMVs selection result already exists. When it receives rewritten sub-global queries lattice from its parents and it has no any children, it just sends its parent the messages including the status of the associated views showing whether the views are materialized, the sizes of associate views, computing costs or maintenance costs, update frequencies of materialized views.
- In our research, we only consider the global view maintenance cost, namely the maintenance cost between GMVs and their source views distributed in the different peers. We ignore the cost of locally maintaining LMVs and the base fact tables and dimensions.
- Each peer publishes its schema, LMVs, and GMVs to its acquaintances.
- Same dimensions in different schemas are the same.
- Uniform, global semantics are assumed.
- Only one instance of P2P GMV selection algorithm executing in the whole system and this instance is the first one in the system. In other words, there are no GMVs selected yet in the system. The case that there are already some GMVs in the system will be studied in the future work.
- All associated sink views of each global query are full views. In other words, each sink view is identical to some element or view in the given local lattice.
- Homogeneous dimension are in all local data warehouses.
- Each peer has only the knowledge of its neighbors, and there exists an engine in each peer to handle the query relevance to its acquaintances.
- Only the views associated with global queries are involved in the P2P global query lattice. We do not consider the common parent views that two or more query views share.
- The global communication costs between peers are ignored except the data transfer cost.
- As for the views maintenance, we use an incremental maintenance technique.

4.2 Distributed GMV Selection Greedy Algorithm

In this section, we describe the distributed GMV selection greedy algorithm, which is used to select sets of materialized aggregate views on different peers to speed up the global queries posted on a given peer in a peer data warehousing system and minimize the total cost of answering queries and maintaining the selected materialized views. We use the diagram in Figure 4.1 to illustrate this algorithm.

The state of a peer is defined as: States S= {Initiator, Idle, Active, Done}

The initiator state of a peer is defined as: S_{init}= {Initiator, Idle}

The termination state of a peer is defined as: S_{term}= {Done}

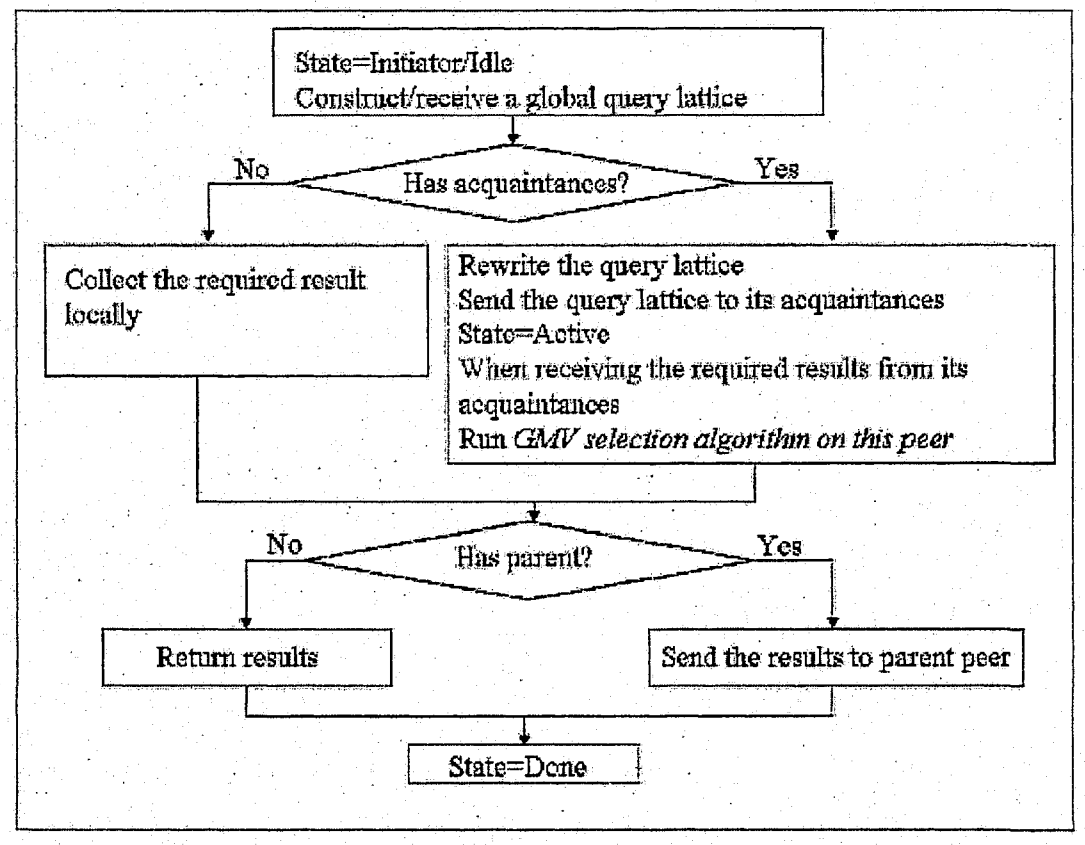


Figure 4.1 P2P distributed GMV selection greedy algorithm flowchart diagram

The detailed P2P distributed GMV selection algorithm in a peer warehousing system is described in the following, while the GMV selection greedy algorithm running on each peer will be discussed in the next section.

SQ-L stands for sub-query lattice

N(x) represents the acquaintances (neighbors) of node x

P2P GMV Selection Greedy Algorithm

Initiator

sent (SQ-L) to N(x)
numOfOutNeighbors := N
become ***Active***

Idle

receiving (SQ-L)
parent := sender
numOfOutNeighbors: = N-1
numOfNoResponse := 0
if (numOfOutNeighbors == 0) *// no more neighbors*
    return the local view selection result
    send "Yes" and the associated results to sender
    become ***Done***
else *//there are neighbors*
    send (rewritten SQ-L) to N(x) –sender
    become ***Active***
endif

P2P GMV Selection Greedy Algorithm (Continued)

```
Active
        receiving (rewritten SQ-L)
        send "No" to sender

        receiving ("Yes" & results)
        numOfOutNeighbors --
        //all neighbors have replied
        if (numOfOutNeighbors == 0)
              calculate the size and query cost of  related query views
              run global view selection algorithm at this peer
              if (parent != null)
                    send "Yes" and the associated results to the sender
              else
                    do nothing
              endif
              become Done
        endif


          receiving ("No")
          numOfOutNeighbors --
          numOfNoResponse ++

           //all neighbors have replied
          if (numOfOutNeighbors == 0)
               if(numOfNoResponse == numOfOutNeighborsForThisNode)
                    return the local view selection result
               else
                    calculate the size and query cost of associated query views
                    run global view selection algorithm at this peer
               endif
               if (parent != null)
                     send "Yes" and the associated results to the sender
               else
                    do nothing
               endif
               become Done
          endif

Done
```

Note: 1. numOfOutNeighborsForThisNode is constant for each node during the period of this algorithm running, we can regard it as one property of the node.

2. The statements in italic style starting with "//" in the algorithms are comments.

4.3 GMV Selection Greedy Algorithm Running on Each Peer

This section describes the GMV selection greedy algorithm running on each peer, which is called from the P2P distributed GMV selection greedy algorithm. Figure 4.2 shows the flowchart diagram of this algorithm.

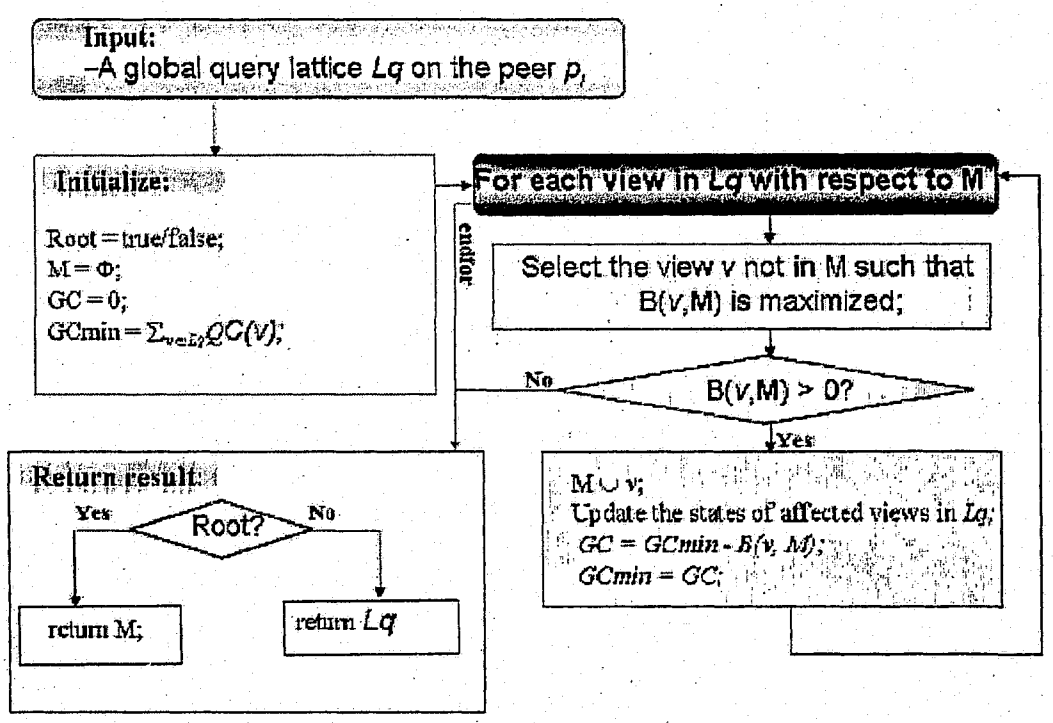


Figure 4.2 The flowchart of the GMV selection greedy algorithm running on each peer

The detailed algorithm is described as follows:

Input:

Given a peer p_i in a P2P data warehousing system G, a set GQ of global queries (each query view has the following attributes: status, size, minimum query evaluation cost, minimum maintenance cost, query frequency, update frequency), a global query lattice L_q on the peer p_i (L_q is the initial global query lattice of the concerned P2P global query lattice or sub-P2P global query lattice).

Output:

- The set M of selected views if this peer is the initial peer, such that the total global answering GQ and maintaining the selected materialized views cost is minimized.
- A set GQ of query views, in which the state of each view will be redefined, if this peer is not the initial peer, such that the total answering GQ and the maintaining the selected materialized views cost on this peer is minimized.

Greedy Algorithm:

begin
//false if the node who calls this alg is not the //root, otherwise, true
root =false/true;
M = ∅ ; *//a set of selected views*
calculate the QC for each view in L_q;

//$QC_A(v)$ is the computing cost of v from its //AND- DAG expression
// GC_{min} is the minimum cost of answering of all // GQs and maintaining all view in M

$GC_{min} = \sum_{v \in L_q} QC_A(v)$;

GC = 0;

Greedy Algorithm: (Continued)

Foreach view in L_q with respect to M
Select that view v not in M such that B(v, M) is maximized;
if (B(v, M) > 0) then
M ∪ v;
Modified the states of affected views in L_q;
GC = GC_{min} - B(v, M);
GC_{min} = GC;
else
goto ***returnResult***;
endif

endfor

returnResult:
if(!root)
return GQ (L_q) with final state for each view;
else
return M;
endif

Note: The statements in italic style starting with "//" in the algorithms are comments.

4.4 Complexity of the P2P GMV Selection Problem

In this section, we discuss the messages complexity and run time complexity of the P2P GMV selection problem, respectively.

4.4.1 Messages complexity

Here we only consider the number of messages but not the sizes of messages. Assume the P2P network is an arbitrary network. The number of links is m. The worst case is that there are 2 messages (sending message and receiving message) on each link when all nodes in the network are relevant to the P2P global aggregate lattice. The process of P2P algorithm executing is in fact also a process of a spanning tree construction. So the total number of messages for the P2P GMV selection algorithm is at most 2m, and therefore the complexity is O(m).

4.4.2 Run time complexity

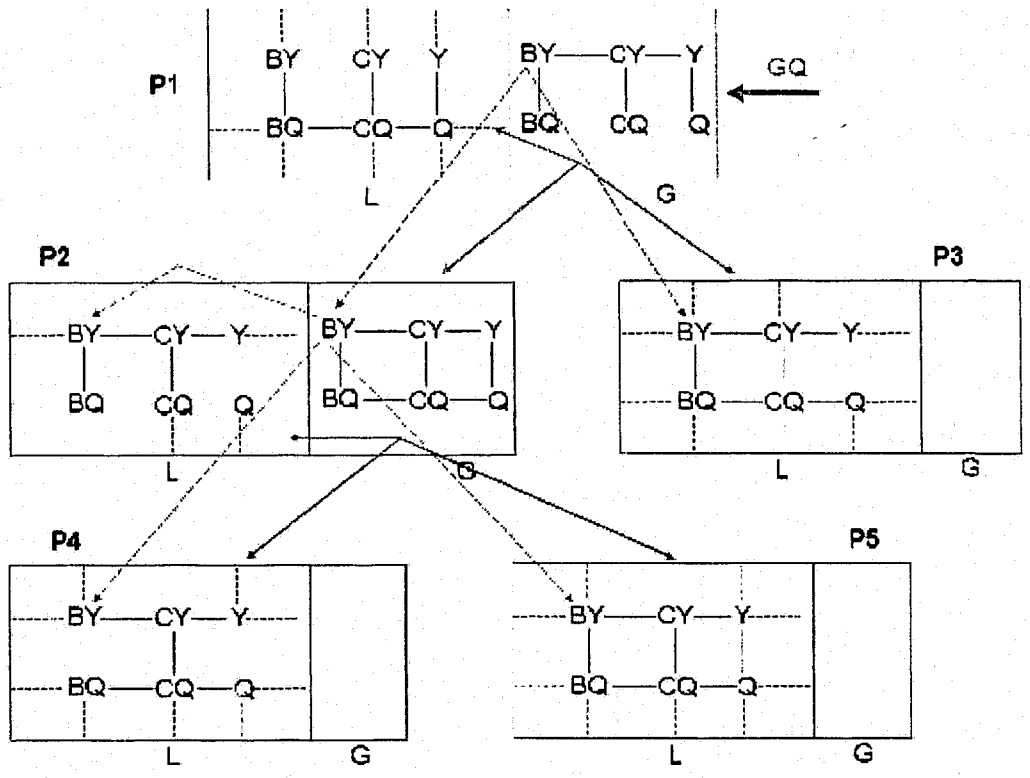
The P2P GMV selection problem is trivially NP-hard, the reasons are as follows:

- The view selection problem for a single data warehouse system under either disk space or maintenance cost constraint is NP-hard [Gupta1997, Gupta1999].
- In P2P data warehousing systems, the complexity of GMV selection running on each node is obviously the same as that of the view selection problem for a single data warehouse. To solve the P2P GMV selection problem, the global GMV selection algorithm will be running at k nodes, so the complexity of P2P GMV selection problem is k times the complexity of the global GMV selection algorithm. Therefore, P2P GMV selection problem is NP- hard.

4.5 Algorithm Running Example

To make the scenario as simple as possible, we assume that the local aggregate lattice and associated view information on each local data warehouse is the same. The view composition of a global lattice is the same as that of the local aggregate lattice. Figure 4.1 illustrates the P2P global aggregate lattice for the walkthrough sample. Table 4.1 displays the information of local views and global views. The size of a view is measured by the number of rows contained in a view. We use 1 to indicate that a view is materialized, and 0

if it is not. f_v and g_v represent the query frequency and updating frequency of a materialized view, respectively.



L --- Local memory partition G --- Global memory partition

Figure 4.3 P2P global aggregate lattice for the running sample

Table 4.1 The information of views in local lattices and global lattices in the experiment system

	Local view		Global view				
View	**size**	**status**	**View**	**size**	**status**	f_v	g_v
BQ	300	1	BQ	300	0	1	4
CQ	80	1	CQ	80	0	2	1
Q	10	0	Q	10	0	1	4
BY	100	1	BY	100	0	1	4
CY	20	0	CY	20	0	1	4
Y	2	0	Y	2	0	1	4

First, let us walkthrough the P2P view selection algorithm:

Table 4.2 The executing process of the P2P view selection algorithm

Initial State	Active State
1. P1(Initiator) Root:=true Sent(SQ-L) to P2, P3 NumOfOutNeighbors:=2 Become ***Active*** **2. P2 (Idle)** Receiving(SQ-L) from P1 Root:=false Parent:=P1 NumOfOutNeighbors:=2 NumOfNoResponse:=0 Send(rewrite SQ-L) to P4, P5 Become ***Active*** **2. P3 (Idle)** Receiving(SQ-L) from P1 Root:=false Parent:=P1 NumOfOutNeighbors:=0 NumOfNoResponse:=0 *//NumOfOutNeighbors==0* *(comment: no more neighbors)* Collect the associated local view selection result Send Yes and the associated results to P1 Become ***Done*** **3. P4 (Idle)** Receiving(SQ-L) Root:=false Parent:=P2 NumOfOutNeighbors:=0 NumOfNoResponse:=0 *//NumOfOutNeighbors==0* *(comment: no more neighbors)* Collect the associated local view selection result Send Yes and the associated results to P2 Become ***Done***	**4. P2 (Active)** Receiving (Yes & results from its children (P4, P5) NumOfOutNeighbors-- *//each time that P2 receives the result from one of its children* if (NumOfOutNeighbors==0) Calculate the size, query cost, predicted maintenance cost of related query views Run global view selection algorithm *//parent!=null* Send yes and the associated results to P1 Become ***Done*** End **5. P1 (Active)** Receiving (Yes & results from its children (P2, P3)) NumOfOutNeighbors-- *//each time that P1 receives the result from one of its children* if (NumOfOutNeighbors==0) Calculate the size and query cost of related query views Run global view selection algorithm *//parent=null* Become ***Done*** End

3. P5 (Idle) Receiving(SQ-L) Root:=false Parent:=P1 NumOfOutNeighbors:=0 NumOfNoResponse:=0 *//NumOfOutNeighbors==0* *(comment: no more neighbors)* Collect the associated local view selection result Send Yes and the associated results to P2 Become ***Done***	

*Note: 1, 2, 3, 4, and 5 in the table represent the node processing order

The statements in italic starting with "//" in the algorithms are comments.

Second, now let us trace the global view selection algorithm on P2.

Step 1: Calculating query costs and maintenance costs of global aggregate views in the rewritten P2P aggregate lattice

When P2 receives the local view selection results from P4 and P5, P2 calculates the query costs and predicted maintenance costs of the global aggregate views in the rewritten P2P global aggregate lattice on P2. Let's suppose that the data transfer rate per unit data is w. *uc_coe* represents the ratio (percentage) of update data size of a view per time to the size of view. Here $w = 1$ and *uc_coe* = 1. P-UC stands for the predicted maintenance cost in case a view is materialized. QC represents the query computing cost.

Here we use BQ and Q as examples to illustrate how to calculate QC and P-UC. In BQ's P2P Expression DAG, all associated views are materialized, so the local computing cost in an acquaintance is the local reading cost of the associated view. The local reading cost from 3 associated views in P2, P4, and P5 is 300*3 = 900, and the data transfer cost from P4 and P5 to P2 is w*300* 2. QC = (w*600+900)* f_v = 1500. Pretended BQ is selected to be materialized, the maintenance cost is data transfer cost * *uc_coe* for one time maintenance.

P-UC = $(w*600*\ uc\_coe)*\ g_v$ = 600 * 4 = 2400. In Q's P2P Expression DAG, its associated views are not materialized in the corresponding lattice. So we have to first find the minimum local computing cost for each associated view Q in the local lattices. Evidently, for each Q in the local lattice can be derived from CQ or BQ. The computing cost from CQ is 80; while the computing cost from BQ is 300, so we select 80 as the minimum cost of computing Q from each local lattice. The local computing cost of Q from 3 associated views in P2, P4, and P5 is 80*3 = 240, and the data transfer cost from P4 and P5 to P2 is $w*10*\ 2 = w*20$. QC = $(w*20+240)*\ f_v$ = 260. Pretended Q is selected to be materialized, the maintenance cost is data transfer cost * *uc_coe* for one time maintenance. P-UC = $(w*20*\ uc\_coe)*\ g_v$ = 20 * 4 = 80.

Similarly, we can calculate all views's QC and P-UC. Table 4.3 shows the calculation results.

Table 4.3 Query costs and predicted maintenance costs of rewritten sub-global queries in the sub-global lattice on P2

View	status	size	f_v	QC	g_v	P-UC
BQ	0	300	1	1500	4	2400
CQ	0	80	2	800	1	160
Q	0	10	1	260	4	80
BY	0	100	1	500	4	800
CY	0	20	1	280	4	160
Y	0	2	1	244	4	16

Step 2: Run the global view selection algorithm on P2

Initialization:

$GC_{min} = \sum QC = 3584$

M=∅

Iteration 1:

1) Selecting a view with the maximum materializing benefit

Table 4.4 The running result for iteration 1

View	Self-benefit	Benefit views	Total benefit
BQ	1500-2400-300=-1200	CQ: 800 – 600 =200 BY: 500 – 300 =200 Other views have no benefit from BQ materializing	-800
CQ	800-160-160=480	Q: 260 -80=180 CY: 280-80=200 Y: 244-80=164	1024
Q	260-80-10=170	Y: 244-10=234	404
BY	500-800-100=-400	CY: 280-100=180 Y: 244-100=144	-76
CY	280-160-20=100	Y: 244-20=224	324
Y	244-16-2=226		226

Note: the views having no benefit from materializing the related view are not listed in the table

From the above table, B(CQ, M) is the maximum, so CQ is selected to be materialized on P2 in iteration 1.

2) Modify the state of aggregate views in the rewritten lattice on P2.

Table 4.5 The new states of aggregate views in the rewritten lattice on P2 after materializing CQ.

View	status	size	f_v	QC	g_v	P-UC
BQ	0	300	1	1500	4	2400
CQ	1	80	2	160	1	160
Q	0	10	1	80	4	0
BY	0	100	1	500	4	800
CY	0	20	1	80	4	0
Y	0	2	1	80	4	0

3) Modify GC and M

M = {CQ}

$GC = GC_{min} - B(CQ, M) = 3584-1024 = 2560$

$GC_{min} = GC$

Iteration 2:

1) Selecting a view with the maximum materializing benefit

Table 4.6 The running result for iteration 1

View	Self-benefit	Benefit views	Total benefit
BQ	1500-2400-300=-1200	BY: 500 – 300 =200	-1000
Q	80-0-10=70	Y: 80-10=70	140
BY	500-800-100=-400		-400
CY	80-0-20=60	Y: 80-20=60	120
Y	80-0-2=78		78

From the above table, B(Q, M) is the maximum, so Q is selected to be materialized on P2 in iteration 2.

2) Modify the state of aggregate views in the rewritten lattice on P2.

Table 4.7 The new states of aggregate views in the rewritten lattice on P2 after materializing Q.

View	status	size	f_v	QC	g_v	P-UC
BQ	0	300	1	1500	4	2400
CQ	1	80	2	160	1	160
Q	1	10	1	10	4	0
BY	0	100	1	500	4	800
CY	0	20	1	80	4	0
Y	0	2	1	10	4	0

3) Modify GC and M

M = {CQ, Q}

$GC = GC_{min} - B(Q, M) = 2560\text{-}140 = 2420$

$GC_{min} = GC$

Similarly, the algorithm will go through iteration 3, 4, 5, 6 or terminate at any time when no more view can be selected. The final result on P2 is M={CQ, Q, CY, Y}, total GC_min=2352. The final states of aggregate views in the rewritten lattice on P2 are displayed in table 4.8.

Table 4.8 The final states of aggregate views in the rewritten lattice on P2

View	status	size	f_v	QC	g_v	P-UC
BQ	0	300	1	1500	4	2400
CQ	1	80	2	160	1	160
Q	1	10	1	10	4	0
BY	0	100	1	500	4	800
CY	1	20	1	20	4	0
Y	1	2	1	2	4	0

Third, let us trace the global view selection algorithm on P1.

Now we calculate the query and maintenance costs of the global aggregate views in the P2P aggregate lattice on P1 after P1 receives the info from P2 and P3. Table 4.9 shows the initial query costs and predicted maintenance costs of global queries on P1. GC_ini at this time is 4976.

Table 4. 9 Query costs and predicted maintenance costs of global queries in the global lattice on P1

View	status	size	f_v	QC	g_v	P-UC
BQ	0	300	1	2700	4	4800
CQ	0	80	2	800	1	320
Q	0	10	1	190	4	80
BY	0	100	1	900	4	1600
CY	0	20	1	220	4	160
Y	0	2	1	166	4	16

After the initial cost has been calculated on P1, we may trace the global view selection algorithm on P1 using the same idea applied on P2. The final selection result on P1 is as follows: M={CQ, Q, CY, Y}. GC_min = 4112. Compared with GC_init =6368 before the P2P view selection algorithm running, the cost is saved 6368-4112=2256.

4.6 Conclusion

In this chapter, we first introduced the assumptions that the P2P GMV selection greedy algorithm are based on. Then we described the distributed view selection greedy algorithm and centralized view selection greedy algorithm on each peer. Following that, we analyzed the complexity of the algorithm in term of both messages and run time context. Finally, we traced the algorithm using our running example.

Chapter 5 Algorithm Implementation and Simulation

In this chapter, we first describe how we implement and simulate the P2P view selection greedy algorithm. Second, we design a series of simulation experiments to find the factors that are affecting the view selection results and the performance of the P2P view selection algorithm according to the experiment results.

5.1 Implementation Setting

In this section, we talk about how we model and implement the P2P view selection problem in a peer data warehousing system.

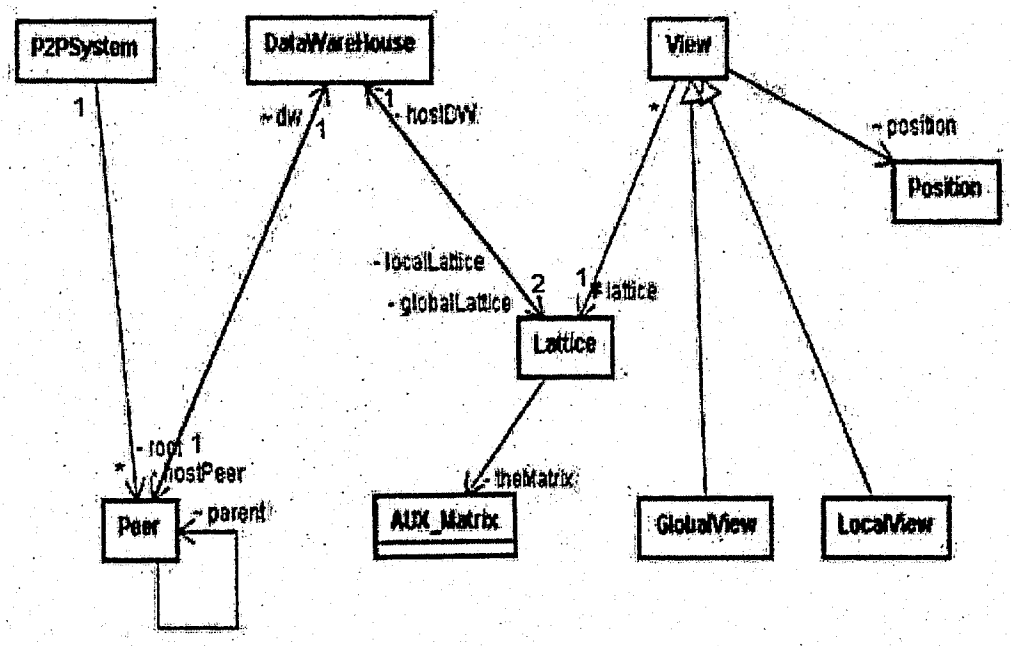


Figure 5.1 Simplified class diagram for the implementation of the P2P aggregate views selection algorithm

We first implement and simulate the P2P aggregate view selection algorithm in a 2-dimension peer data warehousing system using Java programming language. Then we extend the implementation into an n-dimension peer data warehousing system. Figure 5.1 shows the simplified class diagram used to implement the algorithm. The detailed class diagram is displayed in appendix A. From the diagram, it can be seen that there are 8 classes used to implement the P2P view selection algorithm. The *P2PSystem* class encapsulates all the information of a P2P data warehousing system. It is responsible for initializing the system and start executing the P2P view selection algorithm. The *Peer* class represents a peer object in the system. Each peer will hold a data warehouse. The *P2PSystem* consists of a collection of peers. When the P2P view selection algorithm is executed, a spanning tree is constructed among the peers. Each peer acts as a node of the spanning tree. The *DataWarehouse* Class is responsible for encapsulating the information of a data warehouse. Each data warehouse may contain a local lattice and a global lattice. If the peer that holds the data warehouse is a leaf in the constructed spanning tree, this data warehouse contains only a local lattice. The *Lattice* class contains the information of an aggregate lattice. The *View* class has two subclasses: *LocalView* class and *GlobalView* class. The view object represents a view in a lattice. In our implementation, we use a matrix concept to store the views in a lattice. In other words, we use a matrix representing the aggregate lattice in the data warehouse. In the algorithm we need to traverse the whole lattice to find the view with the minimum QC. To realize this objective in the n-dimension implementation, we create a class called *Position*, which is used to store the position information of a view in a lattice. Each view in a lattice is associated with a position in the matrix which represents a lattice. The position of a view in a lattice matrix can be expressed as an array, which contains n elements and each element stands for the position of the view in the corresponding dimension i and i is the index of the element in the array. Also the position of a view can be expressed as a unique sequence number, which can uniquely represent a position in the lattice matrix. When a view is initialized, the position of the view is set. Given a position array, a unique sequence number can be obtained, and vice versa. The conversion rule between the position array and the position sequence number are as follows:

a) Generate the position sequence number *s*

Given a position array a[n] = {a_0, a_1, …, a_{n-1}}, and a dimension length array length[n] = {l_0, l_1, …, l_{n-1}}, which represents the length of each dimension, in other words, the number of views in each dimension, then the position sequence number *s* is calculated as follows:

$$s = \sum_{i=n-1}^{0}(a_i \prod_{j=i-1}^{0} l_j) + a_0$$

b) Generate the position array a

Given a position sequence number s and a dimension length array length[n] = {l_1, l_2, …, l_n}, we have:

a[0] = s % l_1

a[1] = ((s - a[0])/l_1) % l_2

…

a[n-1] = [(s – a[0] –a[1] - … -a[n-2])/l_{n-1}]% l_n

We assume that a view at a "higher" position can be aggregated by a view at a "lower" position. Here if we say a view A is higher than view B, it means A and B are in the same lattice, A and B are not at the same position, and every number in A's position array is bigger than or equal to correspondence in B. For example [3, 4, 2] is higher than [2, 3, 1] or [3, 4, 1] or [3, 0, 2].

All dimensions, lattice matrix information, and experimental data for simulating the algorithm are passed into the program through a text file, called property file. Before executing the algorithm, the program first reads in all experimental data from the property file and then initializes the experimental P2P data warehouse system accordingly. The detailed simulation experiments and results are discussed in the following section.

5.2 Simulation Results and Discussion

Our simulation experiments use the same P2P data warehousing environment as the running example described in Chapter 1. All data warehouses are relational data warehouses and each view in an aggregate lattice is a different SQL query. The only difference among query views in the same lattice is that each query has a different group-by clause. The difference of the same query appearing in different lattices or data warehouses is that their data scopes are different (i.e. the data belongs to different data warehouses). The view in a local lattice only contains the data from the local data warehouse; but to answer the query view in a global lattice, the system collects data from both the local and acquainted data warehouses.

For example, BQ represents the query to obtain the total sales amount of each brand of product in each quarter. Referring to the schema definition of Figure 1.3, the SQL query expression of BQ is:

```
SELECT brandId, qtrId, sum(sales) FROM Sales
GROUP BY brandId, qtrId
```

CY represents the query to obtain the total sales amount of each category of product each year. The SQL query expression of CY is:

```
SELECT name, year, sum(sales) FROM Sales
GROUP BY name, year
```

The data scope of BQ and CY depends on which lattice and data warehouse the BQ and CY are located.

To make things simple, we assume that the local lattices and the global lattices are identical throughout the whole system. Given global queries or a global query lattice as shown in Figure 4.1, Table 4.1 displays that views information in local lattices and global lattices we use in the experiments at most of time. The P2P data warehousing system in the experiment is shown as Figure 1.4. According to the purposes of experiments, we may add more nodes

into the system. Assume that all nodes in the P2P network are relevant to the given global queries. Here we use the number of rows in the query view as the equivalent metrics measure of query cost and maintenance cost.

During the P2P view selection algorithm executing, a spanning tree is constructed. In our experiments, we use the initialization process of query costs of views in the system simulating the process of a spanning tree construction. The height of a tree is the length of the path from the root node to its furthest leaf. In our thesis, the longest path of a P2P network refers to the height of the tree which represents a P2P network. To investigate how the processing time of the P2P greedy algorithm and the average initial and final query costs of P2P global queries are impacted by the longest path of P2P network, the number of granularity levels in the lattice, the number of dimensions of data warehouse, and so forth, we design the following experiments. Each experiment result is the average value of the obtained results by running each experiment setting for 10 times.

Experiment 1:

This experiment is to investigate the effects of the longest path of P2P network and the number of granularity levels in the lattice on the processing time of the P2P greedy algorithm. In this experiment, w, the ratio of unit data transfer cost to unit data reading cost is set to 1; *uc_coe*, the ratio of one time update data size of a view to the size of view, is set to 1. We do a series of experiments for each given number of granularity levels for various longest path of a P2P network. The experiment results are displayed in the Table 5.1 and Figure 5.2. Here d_h stands for the longest path of the P2P network in the experiments, and n_l represents for the number of granularity levels in the lattices. The unit of t_p is nanoseconds.

Table 5.1 The algorithm processing time t_p (ns) at different d_h and n_l

n_l \ d_h	1	2	3	4	5
2	10	20	31	55	65
3	24	40	67	97	122
4	30	65	80	125	135

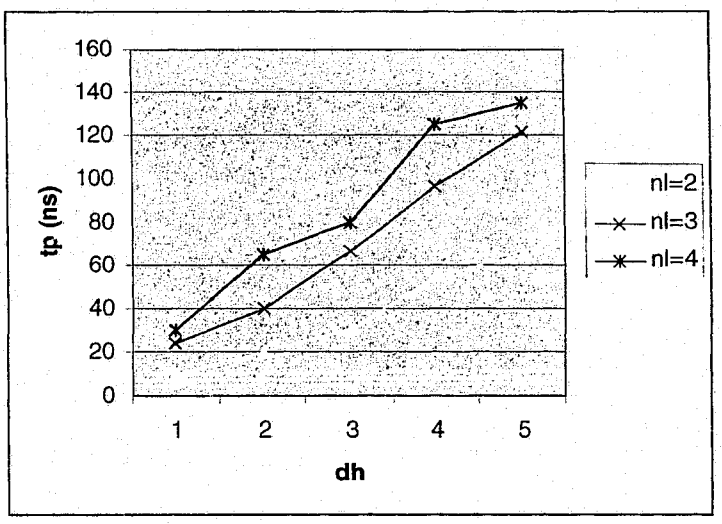


Figure 5.2 This diagram shows how the algorithm processing time t_p changes with d_h and n_l

From the above diagram, we find that the algorithm processing time increases with both the longest path of the P2P network and the number of the granularity levels in the lattices increasing. In our experiment, we ignore the global communication time Δt between nodes. If considering this factor, the algorithm processing time should be revised as the obtained value in the experiment plus Δt * d_h. In this case, the lines in the above diagram would become deeper.

Experiment 2:

The factors of affecting the data transfer cost include network bandwidth, the distance between peers, etc. In the developed algorithm, we calculate the data transfer cost between peers as w * data reading cost for given size of data. w is the ratio of unit data transfer cost to unit data reading cost. To investigate how w affects the average initial query cost and the final minimum query cost of the given P2P global queries as well as the final selection result, we do a series of experiments for various longest path of the P2P network. For each given d_h, we change the value of w. In the experiments, the ratio of materialized view maintenance cost to the local query cost *UC_coe* is 1. In the following tables or figures, *GC_ini* represents the average initial query cost of given global queries in the global query lattice, while *GC_min* stands for the average final minimum combination of query cost and

maintenance cost. Table 5.2 shows the experiments results and Figure 5.3 illustrates the changing trend of *GC_ini* and *GC_min* with *w* for each given d_h.

Table 5.2 GC_ini and GC_min at different d_h and *w*

w / *dh*	0.01	0.1	0.5	1	5
1	401	420	499	597	1387
2	670	706	864	1061	2640
3	938	992	1229	1525	3893
4	1475	1565	1960	2453	6400
5	1743	1851	2325	2917	7653
	104	143	246	326	966

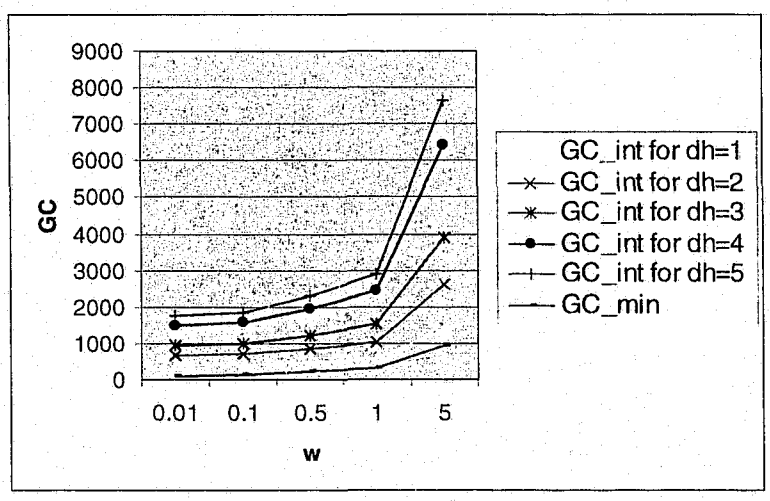


Figure 5.3 This diagram shows the changing trend of GC_ini and GC_min with *w* for each given d_h

From the above diagram, we find that the *GC_min* does not change with d_h changing or we can appropriately say the change of *GC_min* is so small that we can ignored it. However, the *GC_ini* increases obviously with d_h increasing. As a result, in the system with larger d_h, more cost will be saved by materializing some query views. Besides, we also notice that both *GC_ini* and *GC_min* increase with *w* increasing for a given d_h and increase abruptly

after w is larger than 1. This is because the data transfer cost contributes more to the query computing cost and data maintenance cost when the data is transferred from one peer to another peer when w is large. As a result, the view with a large size is not materialized any more due to the large data transfer cost. The final selection results also change with w changing. When w = 0.01 and 0.1, all views are selected to be materialized on P1. When w >= 0.5, BQ and BY are not selected any more.

Experiment 3:

The maintenance cost of a materialized view is another important factor affecting the P2P view selection result. The maintenance cost of a materialized view is determined by its updated frequency and the update cost for each updating. The update cost for each updating is determined by the size of updated data. In this experiment, we only consider the one time update costs of views. We will look at the update frequency in the experiment 4. *uc_coe* represents the ratio (percentage) of one time update data size of a view to the size of view. In our algorithm, we calculate the maintenance cost of one time materialized view as *uc_coe* * the view size. In this experiment, w=1 and d_h=3. The purpose of this experiment is to investigate how *uc_coe* affects *GC_ini* and *GC_min* as well as the final selection result. Table 5.3 records the experiment results and Figure 5.4 demonstrates the changing trend of *GC_ini* and *GC_min* with *uc_coe* changing. The final selection results change with *uc_coe* changing. When *uc_coe* = 0.01 and 0.1, the selected views include BQ, CQ, BY, Q, CY, and Y in the order. When *uc_coe* = 0.5, BQ is not selected; when *uc_coe* =1, BY and BQ are not selected; when *uc_coe* = 5, BQ, BY, and CQ are not selected.

Table 5.3 GC_ini and GC_min at different uc_coe

uc / *GC*	0.01	0.1	0.5	1	5
GC_ini	1525	1525	1525	1525	1525
GC_min	103	140	262	326	392

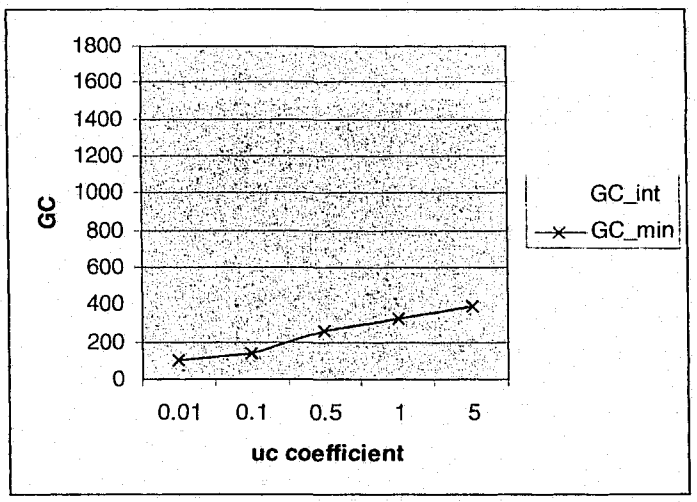


Figure 5.4 This diagram shows how *GC_ini* and *GC_min* change with *uc_coe*

From the above diagram, it can be found that *uc_coe* does not affect *GC_ini*, but it does affect *GC_min*. With *uc_coe* increasing, *GC_min* increases. This result is what we expected because *GC_ini* does not include the maintenance cost and no any global views are materialized at that time. While *GC_min* includes the maintenance costs of materialized views as long as there are some views materialized in the final result. Since *uc_coe* contributes to the final maintenance cost of materialized views, it will affect the final view selection result and this have been proved by our experiment results.

Experiment 4:

The query frequency f_v of global queries obviously has effects on both *GC_ini* and *GC_min*; while the update frequency g_v of materialized views affects *GC_min*. With f_v and g_v increasing, *GC_ini* and/or *GC_min* increase. This experiment is to investigate how query frequency of global queries and update frequency of materialized views affect the final selection result. From a series experiments, we find that with the same ratio of query frequency to data updating frequency and the ratio is smaller than 1/3, the view with a small size has more possibility of being selected. This is because its maintenance cost is lower compared with the view with the large size. When changing the query frequency and/or the

update frequency, the final selection result will be changed in both selected views and selected order.

Experiment 5:

This experiment is to investigate the effects of the number of dimensions of the data warehouse and the number of granularity levels in the lattice on the processing time of the P2P view selection greedy algorithm. In this experiment, w=1, uc_coe=1. We do a series of experiments for each given number of granularity levels for various dimensions of data warehouses. The experiment results are displayed in the Table 5.4 and Figure 5.5. Here n_d stands for the number of dimension of a data warehouse in the experiments, and n_l represents for the number of granularity levels in the lattices. The unit of t_p is nanoseconds.

Table 5.4 The algorithm processing time t_p (ns) at different n_d and n_l

n_l \ n_d	1	2	3	4	5
2	60	50	120	241	1242
3	60	100	240	391	4096
4	60	160	320	1102	10815

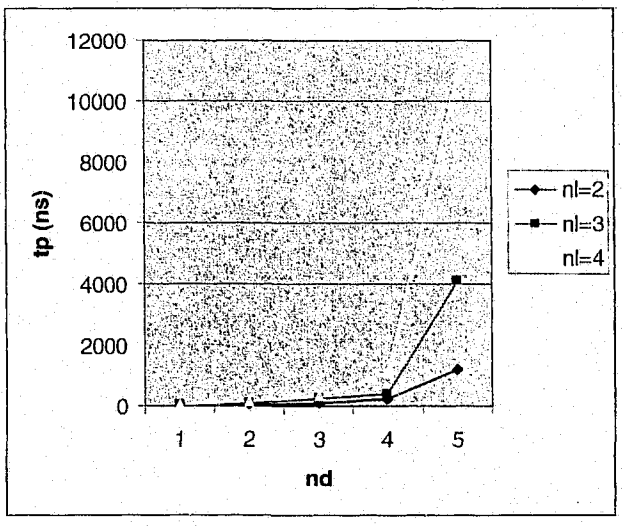


Figure 5.5 This diagram shows how t_p change with n_l and n_d

From the above diagram, the general trend is that the algorithm processing time increases with the increase of the number of dimensions and the number of granularity levels. When the number of dimensions is smaller than 4, the differences of the algorithm processing time for different given numbers of granularity levels are not very obvious. However, when the number of dimensions is larger than 4, the algorithm processing time increases abruptly for each given number of granularity levels and the processing time increase more if the number of granularity levels become bigger.

In this chapter, we have described the implementation setting of the P2P view selection algorithm. After that, we designed a series of simulation experiments to find the factors affecting the final selection result and the algorithm processing time. Through analyzing the obtained experiment results, we found that the longest path of the P2P network, the number of granularity levels in the global lattice, and the number of dimensions in data warehouses affect the processing time of P2P view selection algorithm and the minimum total cost of answering the given global queries and maintaining the materialized views. Moreover, the query frequency, update frequency of materialized view, the data transfer rate, and one time update cost rate of a materialized view have critical impacts on the final view selection results.

Chapter 6 Conclusion and Future Work

In this thesis, we have discussed the core issue that arises in design of a peer data warehousing system, selecting optimal sets of materialized aggregate views on different peers to speed up the global queries posted on a given peer. The view selection problem in a peer data warehousing design is more difficult than that in a centralized data warehouse design, because the costs of global communication, data transfer, and data transformation between peers have to be additionally considered for getting the final integrated global query answers. The P2P view selection problem is an NP-hard problem. This thesis has made a significant contribution in solving the P2P view selection problem.

First of all, we developed a theoretical framework for analyzing and solving the P2P global materialized view selection problem. We categorize materialized views into two categories: LMVs and GMVs; accordingly, we partition the memory of each peer into two parts: local memory storing LMVs, local fact table and dimensions and global memory storing GMVs. We extended the concepts of *Expression AND-DAG* and *aggregate lattice* defined on a centralized data warehouse to express P2P data warehousing semantics. In our problem scope, *P2P Expression AND-DAG* and *P2P aggregate lattice* are dynamically built during the rewritten queries propagation. We also extended the cost model [Gupta1997, Gupta1999] to capture the P2P view selection problem context. In the extended cost model, we took data transfer costs between peers as well as global maintenance costs of materialized views into consideration.

Second, we developed the P2P view selection greedy algorithm to solve the P2P view selection problem. The view selection algorithm is greedy on both distributed base for the whole system and centralized base for each peer. Our work also provides a solution to one of the open problems raised by [Espil2004], how to store and maintain the redundant views with aggregation in peer data warehouses?

Finally, we implemented and simulated the P2P view selection algorithm using object oriented design and object oriented programming paradigm. The simulation experiment

results show that the query frequencies and the update frequencies of materialized views, as well as the data transfer rate have critical impacts on the final view selection results. Moreover, the longest path of the P2P network, the number of granularity levels in the global dimension lattice, and the number of data warehouse dimensions significantly affect the P2P view selection algorithm processing time and the average minimum combination cost of answering the global queries posted on a given peer and maintaining the materialized views. With the longest path of the P2P network, the number of granularity levels, and the number of data warehouse dimensions increasing, the algorithm processing time and the average minimum combination cost of queries evaluation and materialized view maintenance increase.

Our current work is based on the following assumptions:

- The same peer dimension has the same dimension hierarchy and the same category in the same hierarchy is composed of the same set of members.
- Each peer dimension is a homogeneous dimension.
- Only one instance of P2P GMV selection algorithm is executing in the whole system at one time and this instance is the first one of the system.
- The incremental view maintenance technique is used to maintain GMVs whenever the relevant data in fact peers, LMVs, and other associated GMVs is updated.

Therefore, our solution has some restrictions. Our future work will focus on:

- Extending our current solution to the situation that there are already some GMVs in the system.
- Considering the situation that the dimension hierarchy of peer dimension in different peers might be different and the members in each category might not be completely consistent. So schema and instance mappings are needed among peers and as a result data transformation cost should be counted into the cost model.

- Implementing the P2P greedy algorithm in the real P2P context and a P2P architecture using web services.
- Revising the algorithm with the consideration of relevant views or/and aggregate query containment in the global aggregate query lattice.

Bibliography

[AAF] "About Aggregate Functions", http://www.basenow.com/help/About_aggregate_functions.asp

[Baralis1997] E. Baralis, S. Paraboschi and E. Teniente, "Materialized View Selection in A Multidimensional Database", Proceedings of the International Conference on Very Large Data Bases, pp 156-165, 1997.

[Bauer2003] A. Bauer, W. Lehner, "On Solving the View Selection Problems in Distributed Data Warehouse Architectures", Proceedings of the 15th International Conference in Scientific and Statistical Database Management, 2003.

[Bertossi2005] L. Bertossi, "Introduction to Data Integration lecture notes", Carleton University, Ottawa, Canada, 2005. www.scs.carleton.ca/~bertossi.

[Chaudhuri1997] S. Chaudhuri and U. Dayal, "An Overview of Data Warehousing and OLAP Technology", ACM SIGMOD Record 26(1), 1997.

[Cohen2005] S. Cohen, "Containment of Aggregate Queries", SIGMOD record, Vol.34, No.1, March 2005.

[Deshpande1998] P. Deshpande, K. Ramasamy, A. Shukla, J. Naughton, "Caching Multidimensional Queries Using Chunks", 27th International Conference on the Management of Data, SIGMOD'98, Seattle, USA, June 2-4, 1998.

[Espil2004] M. M. Espil, A. A. Vaisman, "Aggregate Queries in Peer-to-Peer OLAP", DOLAP'04, Nov.12-13, pp102-111, 2004, Washington DC, USA.

[Garcia-Molina1999] H. Garcia-Molina, W. J. Labio, J. L. Wiener, Y. Zhuge. "Distributed and Parallel Computing Issues in Data Warehousing", Proceedings of ACM Principles of Distributed Computing Conference, 1999.

[Gribble2001] S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu, "What Can Database Do for peer-to-peer?", WebDB 2001: 31-36.

[Gou2006] G. Gou, J. Yu, and H. Lu, "A* Search: An Efficient and Flexible Approach to Materialized View Selection", IEEE Transactions on Systems, Man, and

Cybernetics---part C: Applications and Reviews, Vol.36, No.3, pp 411-425, May 2006.

[Gupta1995] A. Gupta, I. S. Mumick, "Maintenance of Materialized Views: Problems, Techniques, and Applications", IEEE Data Eng. Bulletin, Special issue on Materialized Views and Data Warehousing, 18(2):3-18, 1995.

[Gupta1997] H. Gupta, "Selection of Views to Materialize in a Data Warehouse", Proceedings of the Intl. Conf. on Database Theory, Delphi, Greece, 1997.

[Gupta1999] H. Gupta, I. S. Mumick, "Selection of Views to Materialize under a Maintenance Cost Constraint", Proceedings of the 7th International Conference on Data Theory, 1999.

[Harinarayan1996] V. Harinarayan, A. Rajaraman, J. D. Ullman, "Implementing Data Cubes Efficiently", SIGMOD, 1996.

[Hurtado1999] C. Hurtado, A. Mendelzon, and A. Abisman, "Updating OLAP Dimensions", Proceedings of the 2nd DOLAP International Workshop, pp 60–66, 1999.

[Inmon2001] W.H. Inmon, "Building the Data Warehouse", Third Edition by Wiley Publishing, Inc., 2001.

[Jarke] M. Jarke, M. Lenzerini, Y. Vassiliou, P. Vassiliadis, "Fundamentals of Data Warehouse", Second, revised, and extended edition, Springer.

[Kawaguchi1997] A. Kawaguchi, D. Lieuwen, I. Mumick, D. Quass and K. Ross, "Concurrency Control Theory for Deferred Materialized Views", Proceedings of the International Conference on Database Theory, Athens, Greece, 1997.

[Kimbal1998] R. Kimbal, L. Reeves, "The Data Warehouse Lifecycle Toolkit", Addison Wesley, 1998.

[Kotidis1999] Y. Kotidis, N. Roussopoulos, A. DynaMat, "A Dynamic View Management System, for Data Warehouses". Proceedings ACM International Conference on Management of Data. SIGMOD'99, Philadephia, U.S.A., June1-3, 1999.

[Martyn2004] T. Martyn, "Reconsidering Multi-Dimension Shemas", SIGMOD record, Vol.33, No.1, March 2004.

[Moeller2001] R. A. Moeller, "Distributed Data Warehousing Using Web Technology", AMACOM, 2001.

[Mumick1997] I. Mumick, D. Quass and B. Mumick, "Maintenance of Data Cubes and Summary Tables in a Warehouse". Proceedings of the ACM SIGMOD Conference, Tuscon, Arizona, pp 100-111, 1997.

[Pedersen] T. B. Pedersen, C. S. Jensen, "Multidimensional Database Technology", Aalborg University.

[Sellies1998] T. Sellies, "Multiple Query Optimization". ACM Transactions on Database Systems, 13(1), 1988.

[Shukla1998] A. Shukla, P. Deshpande, and J. Naughton, "Materialized View Selection for Multidimensional Datasets", Proceeding s of the 24th VLDB conference, pp 488-499, New York, USA, 1998.

[Valluri2003] S. Valluri, S. Vadapalli, and K. Karlapalem, "View Relevance Driven Materialized View Selection in Data Warehousing Environment". The 13th Australasian Database Conference (ADC2003), Melbourne, Australia.

[Yang1997] J. Yang, K. Karlapalem, and Q. Li, "Algorithms for Materialized View Design in Data Warehousing Environment", pp 136-145, In Proc. Of the VLDB, 1997.

[Zhuge1995] Y. Zhuge, H. Garcia-Molina, J. hammer, and J. Widom, "View Maintenance in a Warehousing Environment", Proceedings of the ACM SIgMOD Intl. Conf. In Mngt. Of DATa, pp 316-327, 1995.