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

Md. Delwar Hossain
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

Master of Computer Science
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Querying Communities of Interest in Peer Database Networks

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

Sivarama Dandamudi

Paola Flocchini

Gary W. Slater
LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

The Thesis of Md. Delwar Hossain is approved:

_____

_____

_____

Committee Chairperson

University of Ottawa, Canada

QUERYING COMMUNITIES OF INTEREST IN PEER DATABASE NETWORKS

by

Md. Delwar Hossain

THESIS
Submitted in partial fulfillment of the requirements for the
degree of Master of Science in Computer Science

UNIVERSITY OF OTTAWA
July 2005

Your file Votre référence
ISBN: 0-494-11295-6
Our file Notre référence
ISBN: 0-494-11295-6

ACKNOWLEDGMENTS

I am grateful to some people for the successful completion of this research. Without each of them, this work would have been very difficult to accomplish.

First, I would like to thank my thesis Supervisor, Prof. Dr. Kiringa for his support during this work, his continuous guidance, the countless hours he spent discussing ideas or reviewing my work, and all the fruitful thought that laid the foundation of the research presented in this thesis.

I also thank Mujtaba Khambatti, for supplying his materials and answering lot of questions over e-mail in spite of his busy schedule. His idea helped me proper directions to build our research interest.

Thanks to Anastasios Kementsietsidis for providing his thesis materials. I am thankful to the University of Ottawa for providing the computer laboratory. I am also grateful to some of university staff and colleagues for ample enthusiasm towards my work.

Last, I would like to thank my family members for providing me with courage and support during this period.

DEDICATION

In the name of Allah, the most gracious.

To my daughters

Samiha and Raya.

ABSTRACT

Peer-to-Peer (P2P) networks are used primarily for file sharing. P2P user communities are continuing to grow rapidly, but there is no specific query mechanism for communities of interest in peer database networks. Peer databases are linked to each other through acquaintances. These are individual, independently developed databases that contain local data. Many researchers have tackled the problem of query processing in P2P networks. Moreover, researchers have started to investigate community-based querying in peer database networks. The focus of this thesis is to study a query translation mechanism, to develop an algorithm for querying communities of interest in peer database networks, and to implement a prototype of this algorithm.

P2P communities are formed using common claimed items. First, we investigate an existing community formation and discovery algorithm for a text file, as well as an existing query translation mechanism for peer databases. After that, we develop a community-based search algorithm for peer databases. The developed algorithm combines the aforementioned community formation and discovery with the query translation mechanisms.

A prototype has been developed and experimental results are shown. An approach for discovering communities 'on the fly' is introduced and a method for optimizing the community discovery technique is shown.

TABLE OF CONTENTS

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Synopsis

Peer-to-Peer (P2P) computing is defined as the distribution of computer resources and information through direct exchange. It is a tool highly in demand because it does not need dedicated servers. A node is any single computer connected to a network. A node can work as a client as well as a server. Efficient P2P search techniques are proposed in [16, 39]. Moreover, a search mechanism based on the idea of communities of interest is proposed in [Khambatti, 2003]. A database management system (DBMS) is a set of software programs that controls the organization, storage and retrieval of data in a database. It also controls the security and integrity of the database. Meanwhile, DBMSs are being built around the P2P computing paradigm [2,10, 33]. Such DBMSs are called peer DBMSs. However, there is no specific query mechanism for communities of interest in networks of peer DBMSs.

The term "P2P community" comes from common interests. Here, *common interests* mean the same claimed items. "A non-empty set N of peers forms a peer-to-peer community if all peers in N claims the same attributes" [Khambatti, 2003].

The issue is two fold: (1) how to discover and form communities; (2) to apply community-based search algorithms. Khambatti *et al.* [23, 24] describe an algorithm for forming and discovering communities. They mention that extensive computation or

communication on the components of a peer is not required. Community-based querying assumes guidelines on how users will search in an interest-based locality.

Active research is under way regarding P2P database management. Most of this research (e.g., Piazza project, Hyperion project, etc) is still theoretical. However, some of the research provides prototype implementations. The Hyperion project [2] develops algorithms for peer database `management. Peer DBMSs form a network of nodes (peers) which coordinate querying, updating, and sharing of data at run-time. One of the most important features of a network of peer DBMSs is that peers establish and abolish links at run-time. In this context, the set of all peer DBMSs breaks down into multiple communities of interest formed around common, shared attributes. This thesis provides an algorithm for community-based query processing in peer DBMSs. The idea is to exploit the concept of community of interest to enhance query processing in a network of peer DBMSs.

The key assumptions of our solution are: pre-defined threshold values, items/attributes assigned to each node, each node belonging to one pre-determined group (e.g., AOL group, IBM group, etc), and manually created mapping tables.

1.2 Contributions

We consider a community-based search technique for query processing in peer database networks. The main contributions are:

1. To form and discover communities, extending techniques proposed in [Khambatti, 2002].

2. To combine the community formation and discovery algorithms with a query processing algorithm developed in [22].

3. To consider both explicit communities and community discovery 'on the fly'. 'On the fly' refers to execution time.

Our algorithm forms peer communities by using an attribute escalation method which consists in having a peer claim as many attributes as possible. After discovering the formed communities, our algorithm uses a query translation technique to get results from other peers in targeted communities. Our contribution is the development of a query processing algorithm that will translate a query posed against a peer to the vocabulary of all the peers that belong to the same communities of interest. We implement explicit communities. Moreover, we show how to adapt our framework to discovering communities 'on the fly'.

1.3 Organization

This thesis is organized as follows:

Chapter 2 reviews related and background work. Here, we review federated and multi-database systems, as well as work done on peer-to-peer databases systems, P2P search techniques, and previous uses of the concept of *community* (e.g., web communities, community formation and discovery).

Chapter 3 discusses our community-based search algorithms. This chapter introduces a pseudo-code describing the algorithms and gives our system architecture. The chapter also explains our approach with a motivating example.

Chapter 4 presents an optimization of Khambatti's community discovery procedures. It also contains our experimental results and discusses how communities may be discovered 'on the fly'. Finally, the chapter discusses experimental results obtained from a prototype that we implemented.

Chapter 5 concludes our work with recommendations for future directions.

Chapter 2

Background

2.1 Database Systems

This section describes federated and multi-database systems, Peer-to-Peer (P2P) database systems, P2P information systems, and data mapping/ integration in P2P systems.

2.1.1 Federated and Multi-database Systems

A federated database system (FDBS) is a collection of cooperating but autonomous component database systems (DBSs) [35]. The software that provides controlled and coordinated manipulation of the component DBSs is called a federated database management system (FDBMS). Federated database systems are mainly of two types. The first one is loosely coupled (e.g. MRDSM) and the second one is tightly coupled. Tightly coupled federated database systems have single federation or multiple federations. Sheth and Larson [35] describe federated database systems that have global schema and satisfy distribution, heterogeneity, and autonomy characteristics. For developing FDBSs, Sheth and Larson introduce a reference architecture that has data, databases, commands, processors, schemas, and mapping components.

Litwin *et al.* [29] review multi-database systems that have no global schema. The participant databases of a multi-database system are inter-operable. A set of (multi) databases for which a multi-database language exists is called a multi-database. The

queries of multi-databases are intended to be reusable and probably even remain valid despite changes in the schema of some of the accessed databases. The multi-database language MSQL (Mini Structured Query Language) introduced in this paper has thirteen different new features. The most important of these features is that multiple autonomous databases are able to cooperate in the processing of multi-database queries.

2.1.2 Peer-to-Peer Databases Systems

Peer-to-peer database (P2P DB) networks are a well-substantiated option for the next generation of multi-database systems. Databases can be kept and controlled in local peers or can be made autonomous for other peers. *Autonomy* in this context means that peers decide themselves how to develop their databases, what DBMS to use, how to store data, etc. Such a database kept on a peer is called a peer database.

In Giunchiglia [10], data coordination for interacting among peer databases is proposed. The authors introduce four basic notions to that end: interest groups, acquaintances, coordination rules, and correspondence rules. Interest groups are sets of nodes that are able to answer queries about a certain topic. Acquaintances are recognizable nodes that a node knows about and that have data that can be used to answer a specific query. Coordination rules specify under what conditions - when, how and where - to propagate queries or updates. Correspondence rules take care of the semantic heterogeneity problem. They are implemented as rewrite rules and are called by coordination rules, in the body of the code implementing their action and condition components.

Arenas *et al.* [2] present an architecture of peer database management systems (PDBMS) that has three main components: an interface (P2P API), a P2P layer, and a database. The P2P API is the user interface to the whole system. The P2P layer is the key component of the whole system and comprises three sub-components: an Acquaintance Manager (AM), a Query Manager (QM), and a Rule Manager (RM). The AM is accountable for semi-automatically establishing acquaintances among PDBMSs. The QM executes two types of queries: local queries and global queries. A local query is executed using only the data in the local peer, while a global query uses the P2P network to complement or reconcile locally-retrieved data with data that reside in other peers. The RM enforces stabilization and consistency policies between peers. It creates two sets of rules. The first set includes ones that manage consistency between the data of two peers. The second set of rules is created at query time, after an acquaintance has been established. Gribble [13] emphasizes the complexities of data placement and retrieval as well as the affecting components of data placement problems.

In [3,34], a data model called Local Relational Model (LRM) is proposed. The LRM assumes that the set of all data in a P2P network consists of local relational databases. Each local relational database has a set of acquaintances that defines the P2P network topology. The LRM layer has four modules: User Interface (UI), Query Manager (QM), Update Manager (UM), and Wrapper. The UI allows a user to define queries, receive results and messages from other nodes, and control other modules of the P2P layer. The QM and the UM are responsible for queries and update propagation. They manage domain relations, coordination formulas, coordination rules, acquaintances, and

interest groups. The Wrapper provides a translation layer between the QM and UM and the local data source.

In [14], a Unified Peer-to-Peer Database Protocol (UPDP) is proposed. The UPDP includes messaging, communication, and network protocol models. The Peer-to-Peer Database Protocol (PDP) has a number of key properties. It can interact with any node topology (e.g. ring, star, graph, and tree) and with multiple P2P response modes. Hoschek [14] emphasizes maintaining and querying dynamic and timely information services, resources, and user communities. He provides abstract and concrete PDP messaging models. He uses QUERY, RECEIVE, INVITE, and CLOSE request messages and a SEND response message in the abstract PDP messaging model. For the concrete PDP messaging model, he uses the abstract PDP messaging model with an extensible exchange protocol. The messages QUERY, RECEIVE, INVITE, CLOSE, SEND of the abstract PDP messaging model are refined in a concrete PDP messaging model. The concrete PDP messaging model contains three request message types, two reply message types, one answer message type, and the ERR error type.

2.1.3 P2P Information Systems

Several P2P infrastructures for file sharing exist, such as Napster, Gnutella, Freenet, Chord, JXTA and Gridella. Several of these infrastructures are described in [1]. However, effort is increasingly being spent on using these P2P infrastructures to build full fledged information systems. Such information systems find application in e-commerce and distributed data management. In this section, we briefly describe some of the early P2P infrastructures and information systems that were built on top of a P2P infrastructure, namely Kademlia, PeerDB, and Chord.

Kademlia [32] uses XOR-based metric topology. It is a P2P <key, value> storage and lookup system. It minimizes the number of configuration messages nodes must send to learn about each other. Kademlia can send a query to any node within an interval, allowing it to select routes based on latency or even send parallel asynchronous queries. To locate nodes near a particular ID, Kademlia uses a single routing algorithm from start to finish.

PeerDB [36] is a peer-to-peer distributed data sharing system that has content-based searching facilities. PeerDB is a database application implemented on top of BestPeer, a generic P2P system designed to serve as a platform on which P2P applications can be developed easily and efficiently.

The problem of locating documents in P2P systems is identified in [5] whose authors propose a P2P file sharing system based on Chord. Hashing is a common technique of speeding up access to a collection of data. It transforms a string of characters into a usually shorter fixed-length value or key that represents the original string. Hashing is used in large databases to speed up searching. Chord is an efficiently distributed lookup system based on consistent hashing. It is not a storage system. It associates keys with nodes rather than with data values.

2.1.4 Data Mapping / Integration in P2P Systems

In [20] the problem of mapping data in P2P systems is described. P2P systems rely on value searches to locate data of interest, but different peers can use different values to recognize the same data. To deal with this, P2P systems may rely on mapping tables which record pairs of corresponding values for search domains that are used in

different peers. The mapping tables are binary tables containing pairs of corresponding identifiers from two different sources and are used for searching data. The authors of [20] mention the semantic and algorithmic issues of using mapping tables and present a language that allows the user to specify mapping tables under different semantics. They also provide a solution to the problems of checking whether a set of mapping tables is consistent and whether a mapping table may be inferred from a set of given mapping tables.

Madhavan *et al.* [31] first present a powerful framework for defining languages for specifying mappings and their related semantics. The authors consider an instance of structure for a language representing mappings between relational data and present sound and complete algorithms for the corresponding inference problems.

Lenzerini [28] mentions a strong connection between query answering in data integration and query answering in database with complete information under constraints. The author discusses several approaches to data integration: LAV (local-as-view), GAV (global-as-view), and GLAV (approach that mixes LAV and GAV). The sources defined in terms of the global schema are called source-centric or local-as-view or LAV. The global schema is defined in terms of the sources called global-schema-centric, or global-as-view, or GAV. Lenzerini also shows how to use the GLAV approach in P2P systems where no global schema is assumed. The differences between GAV, LAV, GLAV, and P2P systems are reflected in the sort of formulae used to capture data integration. Finally, Lenzerini presents several open problems with respect to data integration including P2P data integration, data cleaning, reasoning on queries and views, optimization, etc.

2.2 P2P Query Processing

2.2.1 P2P Search Techniques

P2P search techniques are the most vital part of P2P networks. A number of P2P search techniques are available. This section reviews some of them, namely directed Breadth-First-Search (BFS), modified BFS, iterative deepening, local indices, and Intelligent Search Mechanism (ISM). These are the efficient P2P search techniques in comparison with other existing P2P search techniques. Modified BFS is an extension of the current Gnuttela protocol which allows searching with keywords and is designed to minimize the number of messages that are needed to search the networks. ISM uses the past behavior of the P2P networks to further improve the scalability of the search procedure. It builds a profile of peers and uses that profile to find for each query which peers are likely to answer the query. It then forwards the query to those peers only.

2.2.2 Query Translation and Propagation

Query translation and propagation are described in [6, 22, 30, 40]. Kementsietsidis and Arenas [22] present a new mechanism of query translation in peer database systems using mapping tables. They introduce T-queries (tabular form of data) which are used in their query translation algorithm. This query translation mechanism is used for getting answers from the other peers. The authors also provide algorithms for testing whether a given query is a sound translation of another given query and define the notions of sound and complete translations. Sound translations only retrieve correct answers whereas complete translations retrieve all correct answers and no incorrect answers.

Doucet and Lumineau [30] describe the evolution of communities by introducing community knowledge in the process of query propagation. Here, knowledge is used to select the nodes for doing queries. Knowledge also means a user belonging to a community, or establishing a link between related communities. The authors define communities as a small set of keywords that characterize a field of interest. The query first propagates the relevance-based links. At the same time, the query propagates following inter-community links. Then the query linearly propagates towards relevant nodes for finding relevant resources. This process repeats until the end of the Time to Live (TTL) of the query. TTL is an 8-bit field in the Internet Protocol (IP) header that indicates how many more hops this packet should be allowed to make before being discarded or returned. Hops are a number of nodes the data must pass through before it reaches its destination. This value is used to determine the most efficient route. For authoritative records the TTL is fixed at a specific length.

2.3 P2P Communities

P2P communities are formed around common interest attributes. This section reviews some of the notions of *community* used in the literature, along with related algorithms and search techniques.

2.3.1 Web Communities

Flake et al. [7] describe methods for identifying web communities and develop an approximate discovery method that works well in practice. Both the ideal (the entire web can be used for the calculation) and the approximate methods (the authors use this

method for experiment) are used for identifying web communities; they are then compared. Web communities can be defined as a set of web pages that link in either direction to more web pages in the community than to pages outside of the community. The "Max flow-min cut" theorem shows that the maximum flow of the network is identical to the minimum cut that separates a source written to the community and a target outside of it. If we provide a small number of seed web pages, then a small subset of a community can be identified. For this reason, a method for identifying new seeds named Expectation Maximization (EM) algorithm is introduced. In this method, newly discovered web sites are relabeled as seeds, then by re-crawling from the new seeds a new graph is induced. The process reiterates after returning the maximum flow procedure. A maximum-flow based focused crawler can identify new web communities.

By analyzing the link topology, Gibson *et al.* [9] develop a notion of hyperlinked communities on the WWW. They develop an experimental system with the concepts of Hyperlink Induced Topic Search (HITS) for the purpose of studying communities on the web. The HITS algorithm uses the notions of 'hub' and 'authority'. A 'Hub' is a page that links many pages of the same topic. An 'Authority' is a page that is linked to many other pages of the same topic. The HITS algorithm takes a user's keyword query describing some 'topic' as an input, and gives N most-authoritative sites on that 'topic' as an output. Web communities are extracted through link topology using the HITS algorithm.

Kaykova *et al.* [19] emphasize the formation processes of global P2P networks of web-based resources using a mechanism called OntoShell. The OntoShell mechanism is used in a service-oriented environment where each service component has a semantic

profile in the form of a Resource Description Framework (RDF) graph based on the corresponding ontology. The purpose of the formation of P2P communities is to provide services in term of industrial assets and resource management systems. The service provider puts services on OntoShell and creates special meeting places to find partners to get services.

2.3.2 P2P Community Formation and Discovery

The community formation and discovery techniques are described in [23, 24, 38]. In the approach of Khambatti *et al.* [23], communities are formed using claimed interest attributes in active or passive ways. In the active way, claimed attributes are advertised to the 'neighbour' peer from a given peer; in the passive way, a peer advertises its interest attributes on its website and relies on the active approaches of other peers. The process which is used for forming communities is called attribute escalation. The set of attributes of a peer is divised into personal and claimed attributes. Personal attributes are private to a peer. In the attribute escalation process, a peer moves an attribute from the set of personal attributes to the set of claimed attributes whenever it knows that the attribute is claimed by some other peer. This can be achieved through active communication with other peers. This escalation process runs until the collection of peers is stable. The collection of peers is stable when no more attribute can be escalated. At the time a collection of peers is stable, communities have been formed as the set of peers that share the same sets of claimed attributes. Experiments show that peers stabilize after one level of indirection. Khambatti assumes automatic escalation for forming peer communities.

The attribute escalation algorithm facilitates the formation of the communities but is unconscious of its community memberships. After the attribute escalation process, a peer has the list of its claimed attributes and the subset of the claimed attributes that were escalated. If signatures are equal to the attributes that have been recently escalated, then the peer can be considered a member of the communities.

Definition (Khambatti *et al.* [24]) *Let i be a node and C_i be a set that contains attributes claimed by i. Consider a non-empty set N of nodes. Then the set resulting from the intersection of C_k, for all $k \varepsilon N$ is called a signature of the set N.*

In [24], link weights are used to determine the membership of a peer in a community. Link weight is the weight calculated for each claimed attribute of a peer V based on the number of links from V that can reach, after at most one indirection, other peers that claim the same attribute. Then a community discovery algorithm is proposed where link weights of claimed attributes are calculated, threshold values are assumed, and link weights are compared with the threshold values. The peer will assume membership in the community with claimed attribute set if the link weight of each claimed attribute is greater than a threshold value. The threshold value may be low or high, depending on the number of peers in the communities.

Vassileva [38] proposes an approach for supporting file and service sharing in research groups and groups of learners. She uses a multi-agent system for peer help called Community Gnutella (COMUTELLA) that motivates users to participate in the community in P2P systems. For getting help, a University of Saskatchewan student sends his query through his agent who finds other students who are currently on-line and have expertise in the area related to his query. She forms user communities based on a user's

long-term interests, likelihood to search frequently in the same area, and common patterns of behavior. Vassileva designs user modeling to create user groups based on their common interests. The user model is composed of interests, resources (files or services), and relationships of the user. The users have to participate in a community which is a key to success in a P2P system and it is obviously crucial to motivate users in participating in communities. The author observed several levels of user cooperative participation which are: create service, allow service, facilitate search, allow communication, and perform uncooperative free riding. Free riders do not shoulder their fair share of the costs of their use of a resource, involvement in a project, etc. She assumes the users have long-term interests and are likely to search frequently in the same area at different times.

2.3.3 Community-based Search Techniques

Community-based search techniques have been considered in the context of P2P systems [12, 25, 26, 27, 37].

Khambatti *et al.* [25, 26, 27] consider a community-based search (CBS) technique that allows search operations based on peer content rather than only on filenames. Peer communities are based on the idea of "interest groups." Semantic analyzers can implicitly discover interest attributes, but the system design of Khambatti *et al.* obtains entire interests in an explicit manner. They propose a set of rules to be enforced for new peers to join the P2P systems. A search query has three parts: a) the identity of the peer creating the query, b) the actual query for an item, and c) a list of meta-information that describes the item. In the community-based search, a peer might use interest attributes as meta-information to a query. For processing the query, bloom filter summaries are used to

determine the appropriate communities. A bloom filter summary is a compact data structure that probabilistically represents the elements of a set. In the work of Khambatti *et al.*, the bloom filter summary represents a compact summary of the attributes claimed by the peer members of a particular community and keeps the summary into a vector. There are three approaches for selecting the first community peer where the processing of a query starts. Let P_S be this peer. Then the three approaches are: a) P_S is the closest seer (highly popular peers in a community) of a community whose signature S contains at least one attribute from the meta-information list; b) P_S is the seer in the community with S matching the maximum meta-information attributes; and c) P_S is the seer of a community whose signature S matches the most important attributes from a weighted meta-information list of attributes. Khambatti *et al.* also provide a push-pull gossiping algorithm for information searching within P2P communities. The gossiping (push) phase improves the scalability of the algorithm. The efficiency of the pull phase depends exclusively on the efficiency of searches in the P2P system.

Gnasa *et al.* [12] present the architecture of Virtual Knowledge Communities (VKCs) which are built on Peer search memory (PeerSy), and describe the steps in creating representatives of VKCs from a set of n peers. The peer search memory stores the search results locally in P2P networks to retrieve the web content. An Internet search engine, like Google, executes its general query-process, and activates the PeerSy database which sends out queries to the peer networks and matches with the representatives of different VKCs. If the query-process does not match any VKCs, then it finds out VKCs through a Seldom Asked Queries (SAQ) list. If no match is found, the query-link is inserted into the SAQ-list.

Sripanidkulchai *et al.* [37] propose interest-based shortcuts to link peers that share similar interests closer together. In this way, peers can benefit through direct cooperation. This technique is used on top of the Gnutella network to enhance the performance of P2P systems. The creation of shortcuts is based on the Gnutella topology, which depends on flooding queries to all peers. For the first few searches, the peer uses the underlying Gnutella topology. One of the return peers is selected from random and added to the shortcut lists. The subsequent queries go through the shortcuts lists first. If shortcut lists fails, it issues a lookup through the underlying Gnutella topology and repeats the process for adding new shortcuts. Each peer allocates a fixed-size amount of storage which limits to 10, and removes the low utility shortcut link when the list is full. Each shortcut is ordered by many metrics, such as success rate, path latency, load characteristics (query packets per peers process in the system), query scope (the fraction of peers in each query), and additional state kept in each node. The useful shortcut ranks at the top of the list, and an improving rank is determined by an increasing success rate. This technique uses exact matches for finding content locations in a peer. Peers send queries sequentially to each peer in the shortcut structure until the content is found. The shortcut structure is considered as a directed graph where vertices are peers and edges are shortcut relationships. In the shortcut structure, peers are grouped based on interests. Peers store performance history for all of their shortcuts due to the restriction of storage capacity.

2.4 Summary and Conclusions

This chapter presented methods for forming and discovering communities, as well as community-based search techniques which are directly related to our work. Next it

presented some approaches to web based communities, P2P search techniques, P2P information systems, peer-to-peer databases, query translation using mapping tables, and federated and multi-database systems.

Our work introduces a novel community-based search in peer database networks. Our technique is closely related to the work of Khambatti. For this reason, we mention the following shortcomings of his work. His community formation, discovery, and search techniques are based on text files. Though his work does not relate to peer databases, he has given concrete ideas regarding P2P community formation and discovery. He investigates randomly created communities, but does not clearly describe the creation of them. The community discovery algorithm takes several iterations to discover the membership of a community. In Khambatti's community-based search technique, there are three approaches for selecting where to start the search. For the third approach, the criteria for determining the most important attributes as well as the ordering/sorting process of the meta-information of a query item are not clearly described. This community-based search technique is applicable to specific applications like digital libraries, where common interests are well-defined and understood by peer members. Sometimes communities will not form using the mentioned techniques due to a lack of matching common interests.

Our community formation, discovery, and search algorithms are based on peer databases instead of text files. We did not create communities randomly; rather communities are formed based on real commonly claimed attributes. Our algorithm reduces extra iterations for discovering communities. Our techniques extract meta-information easily from the condition parts of a given query. If a query has no condition

part then query considers all attributes of database as meta-information. We did not mention the format of interests of peers. Our solution overcomes most of the shortcomings above and provides an algorithm for community-based search in peer database networks.

Chapter 3

Querying Communities of Interest

3.1 Community-based Querying Method

The main ingredients of our community-based query processing approach are queries, communities, and peer databases. This section describes the methodology of community-based query with a running example, the pseudo-code of our algorithm, and the architecture of our community-based search. It also illustrates all the terms that are used in this chapter.

3.1.1 Basic Definitions

Before dealing with the community-based querying method, the following definitions should prove useful.

1 Peer-to-peer network: *A peer-to-peer network is a network where there is no central server (computer) controlling the network or hierarchy among the computers. It relies on computing power at the edges (ends) of a connection rather than in the network itself.*

2 Peer database network: *A peer database network is a peer-to-peer network where each peer contains databases locally and shares these databases among themselves.*

3 Flat file: *A flat file is a code file that contains generally two columns, one for the codes, and the another one for the description of the codes. It is a database table.*

4 Main file: *A main file is a master file where all necessary information is kept. It contains data of some or all of the attributes as a code for security purposes. It is a main table of the database.*

5 T-query: *A T-query is a tabular form of data. It is used in a query translation algorithm to get answers from other peers. It translates a given query using join operation (OR) for each row.*

6 ItemFile: *ItemFile is used to store the description of attributes of a main file. More specifically, it provides the full name of each attribute of the main table of a database.*

7 Database graph: *A Database graph is a graphical and relational representation of different tables of a database.*

8 CommunityBoard: *A CommunityBoard is a data structure that stores community peers identity (e.g., peer name, signature of claimed attributes set).*

9 Attribute escalation: *This is a process in which attributes are promoted from a private attribute of a peer to a claimed attribute set to form communities.*

10 Mapping rule: *This is a rule that is imposed for data mapping in peer-to-peer data sharing systems.*

11 Mapping Expression: *The following is an inductive definition of mapping expression.*

- *Every mapping μ is a mapping expression*
- *If μ be a mapping expression then $\neg \mu$ is also a mapping expression*
- *If μ_1 and μ_2 be two mapping expressions then $\mu_1 \wedge \mu_2$ and $\mu_1 \vee \mu_2$ are also mapping expressions*

12 *A set of attributes to match a meta-information of a given query means the attributes of the conditional part or attributes of a given query is similar to a set of attributes.*

Our solution has two steps:

1. Forming and discovering communities.
2. Translating a given query to be sent to the desired communities.

Step 1: Communities are discovered through database attributes/items. The attributes of peer databases are linked with ItemFile. Each peer claims some attributes from ItemFile then calculates the link weight of these claimed attributes. After that, it compares this link weight with a globally predetermined threshold value. Calculation of this threshold value is based on a number of observations. This is a "magic number" that is unscientifically assigned. If the link weight is greater than the threshold value then the peer belongs to a community with those attributes as signature. Each peer may belong to one or more communities. At the end of Step 1, every peer knows about communities to which it belongs and which attributes it claims.

Step 2: The next task is to translate the given query in order to search the desired communities. Each peer determines to which communities to send the query by checking attributes of the meta-information list of a given query with claimed attributes of a given peer in community. Meta-information is the conditional part of an SQL query. First, the querying peer searches for a peer where the processing of a given query starts. Let Ps be this peer. The querying peer searches Ps arbitrarily. Mujtaba [27] proposes three approaches for finding Ps, but our solution does not need those approaches because in peer databases we are using SQL queries whose results will be accurate and exact at all times. In the context of text files, results are not exact at all times. Partial results are also accepted. The given query is copied into a CommunityBoard. Periodically and asynchronously each neighboring peer visits the CommunityBoard, and if it can answer

the given query, then it sends results to the querying peer. The visiting neighbor peer also copies the given query into its own CommunityBoard. This process repeats until all peers of the community have been checked. The given peer uses mapping tables to translate the query. If mapping tables linking the query peer with the target peer are empty, then the peer uses direct peer databases for translating the query. Mapping tables between the set of attributes are given in each peer. The given query is translated only for those peers that belong to the desired community.

3.2 Motivating Example of Community-based Search

Consider ten peers P_1, ... , P_{10} of movie communities as our example. Each peer has personal and claimed attributes. Assume that the querying peer is P_1. Suppose that peer sends an "Action related movie" query, which is denoted as P_Q. We assume that the ID of the peer P is X, and that the desired query is in SQL. Moreover, the meta-information of the query P_Q is "year - 2003", "Type - Action", and "CountryName - America". Figure 3.7 illustrates the movie domain that we use as a running example.

3.2.1 Peer Schemas and Database Graph

A peer schema is the local database that mentions attributes of the main file and all flat files. Each peer contains one or more databases. In our example, we are showing the aforementioned movie schema. To be brief, we here describe only peers P_1, P_7, and P_{10}.

Peer P_1 Schema:

Movie (MovieID, DirName, TypeID, CountryCode, AwardType, ActName, Year)
MovieName (MovieID, MovieTitle)
MovieType (TypeID, Type)
Director (DirName, CountryCode, MovieID)

Country (CountryCode, CountryName, MovieID)

Award (AwardType, AwardName, MovieID)

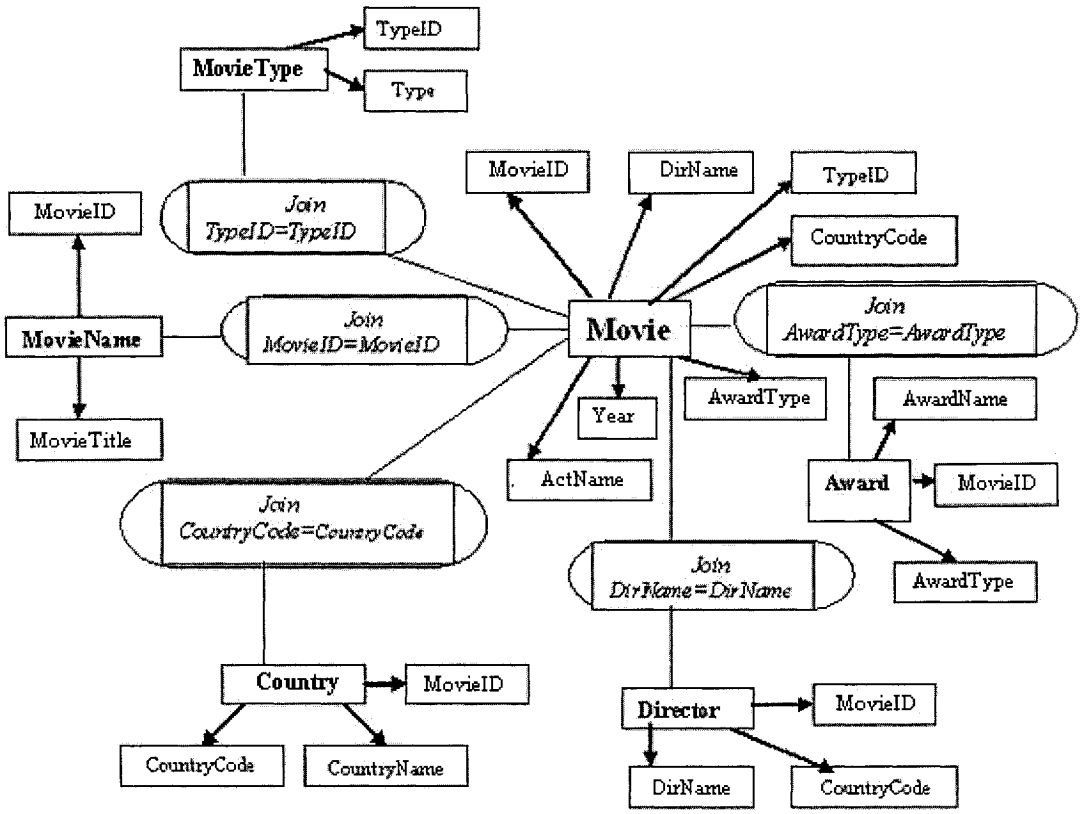

Figure 3.1: Database Graph of Peer P_1

A database graph (e.g. Figure 3.1) shows the relationship between the main file and flat files. More specifically, it shows join attributes between the main file and flat files.

Peer P_7 Schema:

MovieDesc (MovID, Country, MovCategory, AwardCategory, ActorName, ActressName, MovieYear)

Movie (MovID, MovName)

Category (MovCategory, CategoryName)

Award (AwardCategory, AwardTitle)

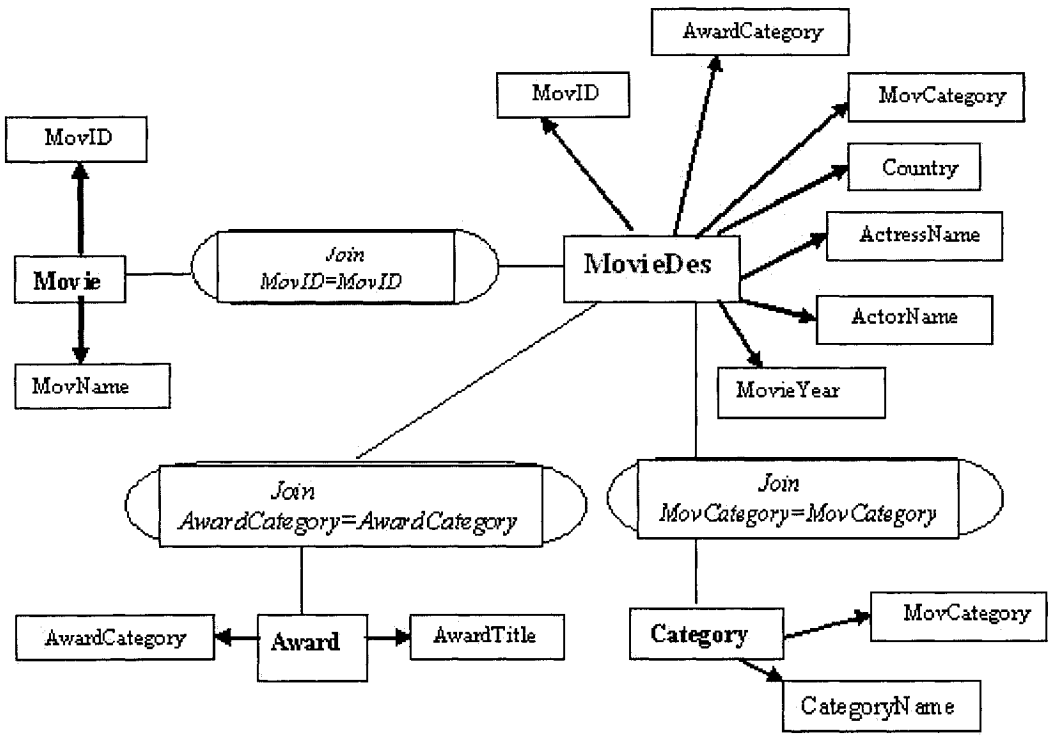

Figure 3.2: Database Graph of Peer P_7

Peer P_1 and P_7 contain the same attributes but with different names. From Figures 3.1 and 3.2, and the respective database schema, we see that P_1 has five flat files whereas P_7 has three flat files.

Peer P_{10} Schema:

Film (MovieName, MovieType, CountryOfMovie, MovieAward, ActorActress, YearOfMovie)

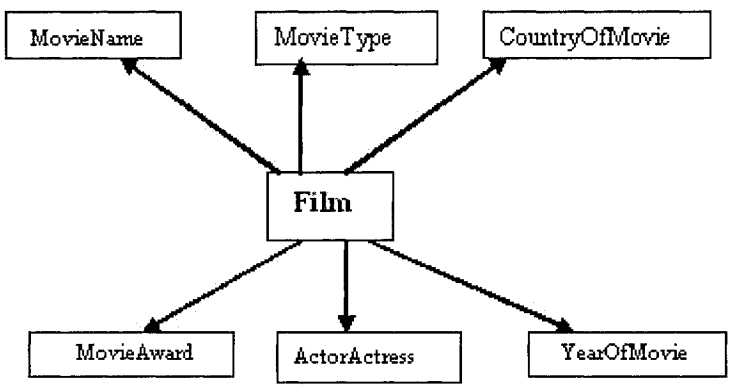

Figure 3.3: Database Graph of Peer P_{10}

From Figure 3.3 and the P_{10} database schema, we see that P_{10} does not contain all the attributes that P_1 and P_7 contain. It is worth mentioning that P_{10} does not have any flat files. The remaining peers also contain similar movie schemas.

3.2.2 ItemFile of Schemas

An ItemFile stores data about data. The ItemFile of our example describes attributes and description of the main file of each peer. Each descriptive item is linked with one attribute of the main schema.

In order to recognize the claimed attributes in every peer, ItemFile is needed. An ItemFile resides in the front-end whereas the database is linked with the ItemFile in the back-end. Peers declare claimed attributes through ItemFile, but the processing (e.g. calculation of link weight, discover communities, etc.) is done by the related database. For simplicity, ItemFile of peer P_1, P_7, and P_{10} are described here, but the rest of the peers contain each likewise an ItemFile with schema link.

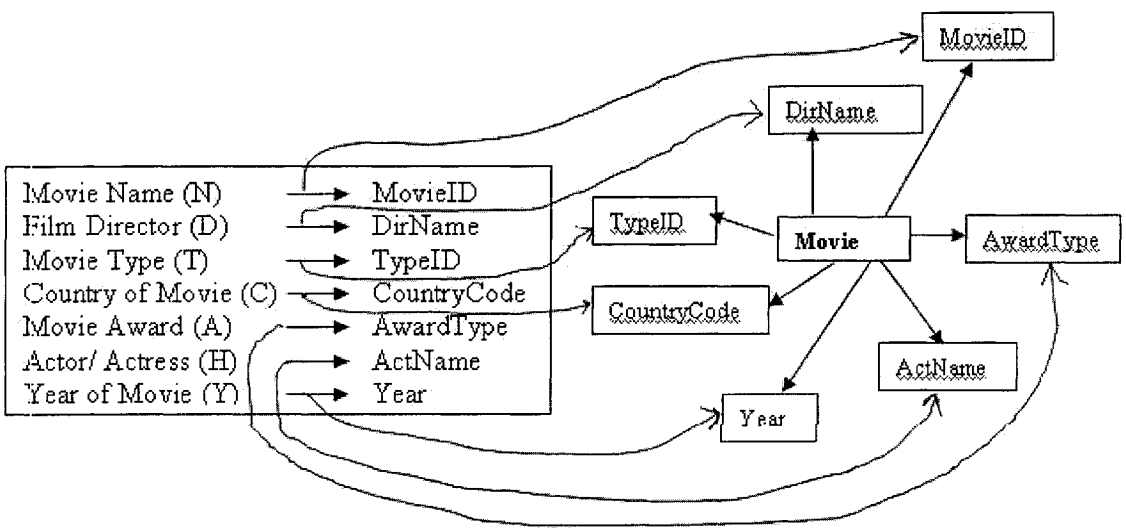

Figure 3.4: ItemFile with Schema Link of Peer P_1.

Figure 3.4 represents the ItemFile of peer P_1 and links with database schema as well. The ItemFile of peer P_1 shows that the "Movie Name (N)" item is linked with the "MovieID" attribute of "Movie" database. The "Film Director (D)" item is linked with "DirName" attribute. Similarly, other items are linked with other attributes of the same database.

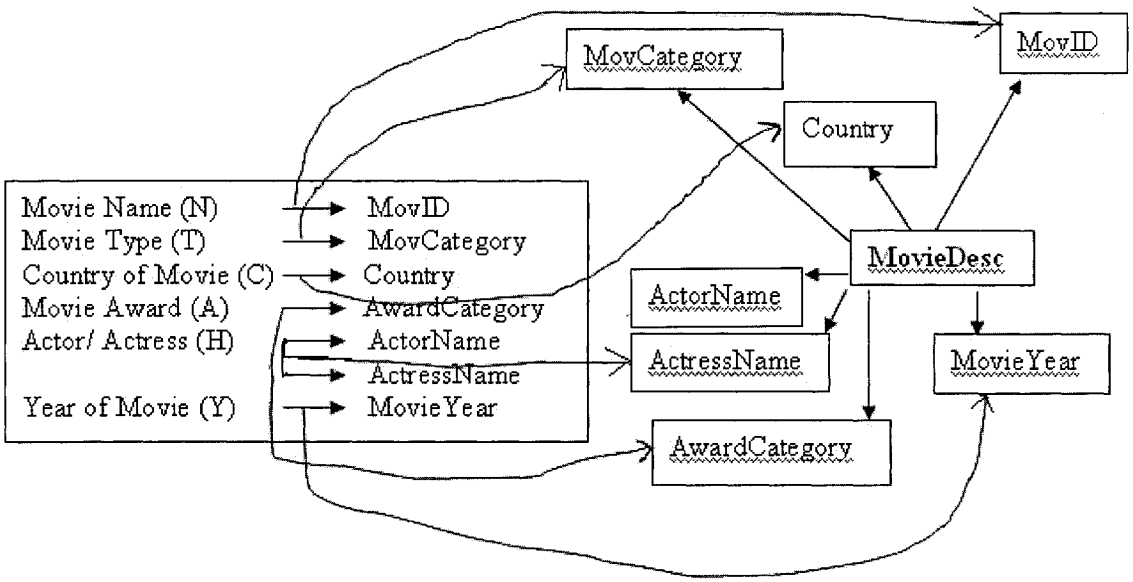

Figure 3.5: ItemFile with Schema Link of Peer P_7.

The ItemFile of peer P_7 in Figure 3.5 shows that the "Movie Name (N)" item is linked with the "MovID" attribute of "MovieDesc" database. The "Movie Type (T)" item is linked with "MovCategory" attribute. In the same way, other items are linked with other attributes of the same database.

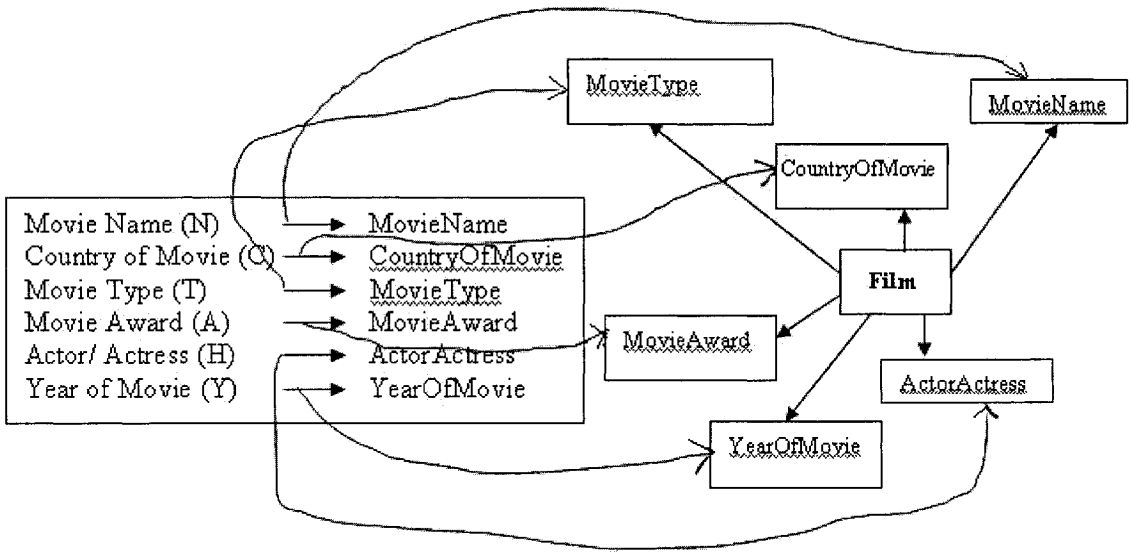

Figure 3.6: ItemFile with Schema Link of Peer P_{10}.

Figure 3.6 illustrates the ItemFile of peer P_{10} and shows its link with the related schema. The figure shows that "Movie Name (N)" item is linked with the "MovieName" attribute of "Film" database. The "Country of Movie (C)" item is linked with "CountryOfMovie" attribute. Likewise, other items are linked with other attributes of the same database.

3.2.3 Attribute Mappings

Different peers may contain the same attributes under different names. Through attribute mappings, schemas of two different peers match with their same attributes. In Table 3.1 and 3.2, only attribute mappings needed for our motivating example are shown. The motivating SQL example query Q is represented in Figure 3.7.

Table 3.1: Attribute Mapping between P_1 and P_7

MovieID	MovID
Year	MovieYear
TypeID	MovCategory
CountryCode	Country

In Table 3.1, MovieID is mapped to the MovID attribute because these two attributes declare the same types of data of peer P_1 and P_7. Similarly, Year is mapped to MovieYear, TypeID is mapped to MovCategory, and CountryCode is mapped to Country.

Table 3.2: Attribute Mapping between P_7 and P_{10}

MovID	MovieName
MovieYear	YearOfMovie
MovCategory	MovieType
Country	CountryOfMovie

In Table 3.2, MovID, MovieYear, MovCategory, and Country of peer P_7 is mapped with the same attributes of peer P_{10}.

3.2.4 Mapping Tables

A mapping table associates values residing in different peers and provides a mechanism for sharing data. Mapping tables and mapping expressions are the basic tools for exchanging information between peers [2]. We use mapping tables for discovering communities in some cases, but they are mainly used for query translation of different peers. Both single and multiple mapping tables are usually used for query translation. In

this example, we use only a single mapping table. The creation and maintenance of mapping tables are done by a database administrator.

Table 3.3: Mapping Table between YearToMovieYear

Year	MovieYear
1982	82
1990	90
1999	99
2001	01
2002	02
2003	03

Table 3.3 shows the data value mapping of the "year" attribute between peer P_1 and P_7. In peer P_1, "year" is mentioned as four digits (e.g. 1982), whereas in peer P_7, the same attribute is declared as two digits (e.g. 82). Through this mapping table, it is easily recognized that year 1982 is equivalent to year 82. Now, Peer P_1 and P_7 can communicate and recognize the year field through mapping table 3.3.

Table 3.4: Mapping Table between MovieYearToYearOfMovie

MovieYear	YearOfMovie
82	1980
90	1990
99	1990
01	2000
02	2000
03	2000

Similarly, Table 3.4 indicates the data value mapping of the "year" attribute between peer P_7 and P_{10}. In peer P_7, "year" is defined as two digits (e.g. 99), but peer P10 is defined as four digits (e.g. 1990's) in terms of decades, not specific years.

Table 3.5: Mapping Table between TypeIDToMovCategory

Type	**CategoryName**
Action	War
Action	Fighting
Drama	Romantic
Drama	Love Story
Horror	Comedy
Horror	Futuristic

Table 3.5 shows the data value mapping of "movie category" between peer P_1 and P_7. In peer P_1, action related movies are defined as "Action," whereas in P_7, the same type of movie is defined as "War" and "Fighting". Similarly, drama movies are declared in peer P1 as "Drama," whereas the same type of movie is declared in peer P7 as "Romantic" and "Love Story."

Table 3.6: Mapping Table between CountryNameToCountry

CountryName	**Country**
America	USA
Canada	CA
England	UK
India	IND

Table 3.7: Mapping Table between CountryToCountryOfMovie

Country	**CountryOfMovie**
USA	North America
CA	North America
UK	Europe
IND	South Asia

In Table 3.6 and 3.7 the values of Country of peers P_1 and P_7, as well as those and of peers P_7 and P_{10} are shown. In peer P_7, country is mentioned in abbreviation (e.g. CA, IND), whereas in peer P_{10}, country is mentioned in continent (e.g. Europe).

3.2.5 Running Example

The movie community contains the following attributes, which are denoted as a letter. In the diagram of Figure 3.7, letters are used instead of full attributes names. The attributes are:

Movie name = N

Film Director = D

Movie type = T

Country of movie = C

Movie award = A

Actor/Actress = H

Year of movie = Y

Each peer has a CommunityBoard, but only the CommunityBoard of the initial Ps (peer P_7) is shown in order to keep the diagram simple. In Figure 3.7, the SQL query, and the meta-information are also shown. Similarly, local database schemas, the database graph and the ItemFile of peer P_1 (querying peer), P_7 (initial Ps), and P_{10} (neighbor peer of Ps) are shown in the corresponding Figure 3.1, Figure 3.4, Figure 3.2, Figure3.5, Figure 3.3, and Figure 3.6.

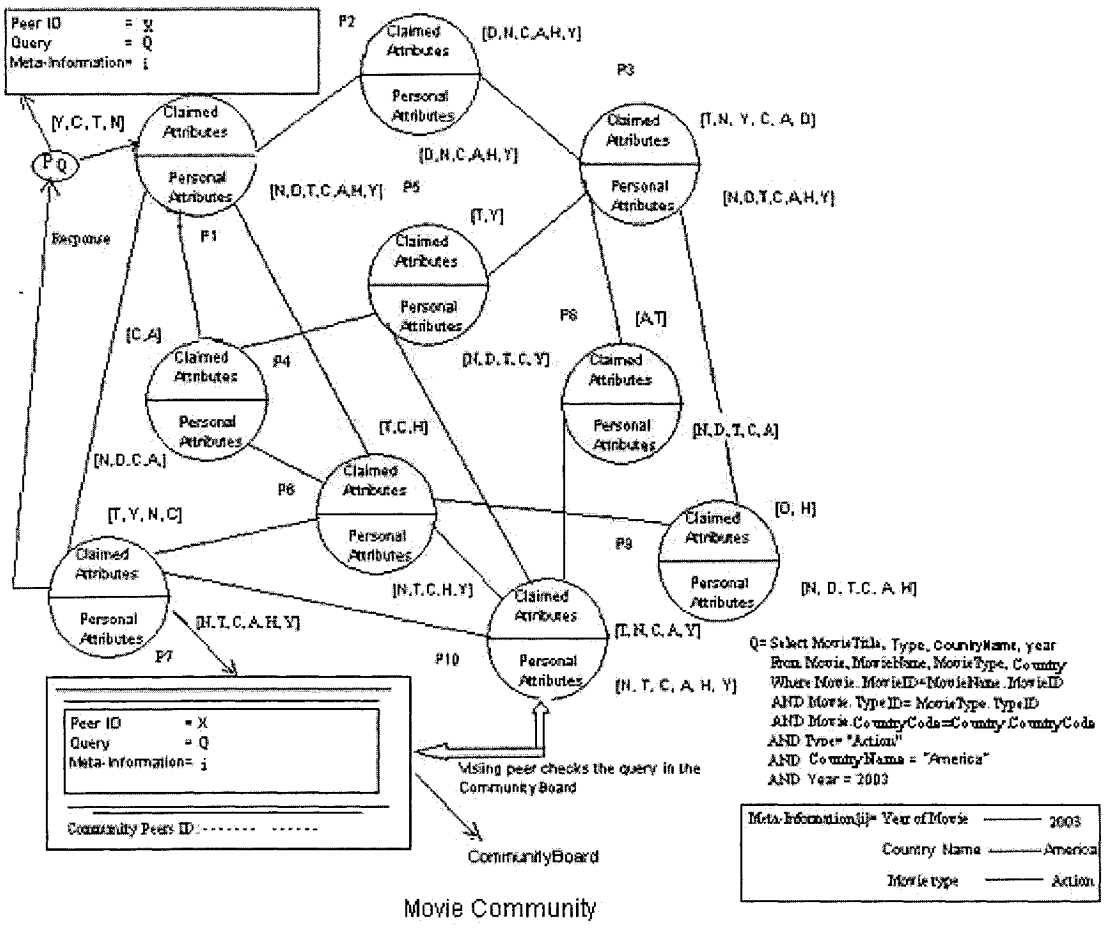

Figure 3.7: Diagram of Movie Community

We assume that communities are already formed using an attribute escalation method where attributes are claimed through the ItemFile of peer databases. Communities are discovered by computing the link weight and comparing the latter with the predetermined threshold values. We set a 60% threshold value in our implementation.

Communities vary with an increase or decrease in threshold value. The claimed attributes list of each peer and the peer identity of communities are created when communities are discovered. Now, the claimed attributes of each peer is checked against the meta-information list of the query to find the initial Ps. The user query is written in SQL and the meta-information is stored in an array which is extracted from the condition

parts of the SQL query. If there is no condition part, then all attributes are considered as meta-information. The Ps stores this SQL into its CommunityBoard, translates this query and sends the results to the querying peer. Other visiting peers (neighbour peers) check the CommunityBoard of Ps and copies this SQL into its own CommunityBoard. The visiting peer sends its result (if any) to the querying peer. All visiting peers of a community check their claimed attributes against the meta-information; if there is a match the query is translated. This process continues until the members of the community have been exhausted.

We use the following running example about movies. Assume that a user wants to retrieve all action related American movies shot in the year 2003 from peer P_1. Then the Query "Q" should be:

Q: Select MovieTitle, Type, CountryName, year

From Movie, MovieName, MovieType, Country

Where Movie. MovieID= MovieName. MovieID

AND Movie. TypeID= MovieType. TypeID

AND Movie. CountryCode= Country. CountryCode

AND Type= "Action"

AND CountryName= "America"

AND Year=2003

To retrieve information from the peer P_7 database, the query should be as follows:

Q': Select MovName, CategoryName, Country, MovieYear

From MovieDesc, Movie, Category

Where MovieDesc. MovID= Movie. MovID

AND MovieDesc. MovCategory= Category. MovCategory

AND (CategoryName= "War" OR CategoryName= "Fighting")

AND Country= "USA"

AND MovieYear=03

To retrieve information from the peer P_{10} database, the query should be as follows:

Q'': Select MovieName, MovieType, CountryOfMovie, YearOfMovie

From Film

Where MovieType= "Action"

AND CountryOfMovie= "North America"

AND YearOfMovie=2000

To get information from the rest of the peers of movie communities, the original query Q should be translated to the vocabulary of the remote peers using the mappings that are in place.

3.2.6 Working Procedure of Our Example

In the context of our running movie community example, we are describing how each query is translated after searching the desired community and getting expected results from the rest of the community's peers. We assume that each peer in the network shares some attributes over the network. Each peer then defines mapping rules and the attribute-mapping table.

Peer P_1 poses SQL query (P_Q) which has three constraints. From this SQL query, we get the following meta-information: year (2003), movie type ("Action"), and country ("America"). Peer P_1 has these claimed attributes, so the user gets results from P_1. Then query P_Q looks for Ps arbitrarily in the same community. As we have claimed attributes

of each peer and the identities of all peers of each community, so P_Q travels within its community. We assume P_Q checks claimed attributes of peer P_7 with the meta-information of the given query. The claimed attributes of peer P_7 matches with the meta-information of a given query, so we can consider peer P_7 as Ps. Peer P_7 then stores this query P_Q in the CommunityBoard and starts a translation process to get results. If a mapping table is found it applies; otherwise it uses mapping rules and attributes of the mapping of the local peer database. The database graph is used to find relations and necessary join operations. We did not show any mapping rules because our example query does not need mapping rules. For query P_Q we need attribute mapping and a mapping table.

For translating the given query, P_Q performs the following tasks:

First, P_Q is converted to its corresponding T-Query (Q_T) which is:

MovieTitle	Type	CountryName	Year
M1	"Action"	"America"	2003

Second, computes T-Query (Q'_T) by joining Q_T with mapping Table 3.5 of peer P_7:

MovName	CategoryName	Country	MovieYear
M1	"War"	"USA"	03
M2	"Fighting"	"USA"	03

Finally, the Q'_T translates the query using a join operation (OR) for each row. We notice that the database graph of Figure 3.2 subsumes attributes MovName, CategoryName, Country, and MovieYear. From the graph of Figure 3.2 we see that we need join relations for MovieDesc, Movie, category, and Country. The complete translated query is as follows:

Q': Select MovName, CategoryName, Country, MovieYear

From MovieDesc, Movie, Category

Where MovieDesc. MovID= Movie. MovID

AND MovieDesc. MovCategory= Category. MovCategory

AND (CategoryName= "War" OR CategoryName= "Fighting")

AND Country= "USA"

AND MovieYear=03

Then the visiting peer P_{10} (neighbor of P_7 or Ps) of the community checks the query into the CommunityBoard of P_7. If there is a match then a translation of the given query is started; otherwise P_{10} copies this query (P_Q) into its own CommunityBoard. The visiting peer of P_{10} (suppose P_6) checks the CommunityBoard of P_{10}. If there is a match then the query is translated and results are sent to the querying peer; otherwise it sends nothing, but copies the query into its own CommunityBoard. This procedure repeats for all of members in the community.

3.3 Algorithm for Community-based Search

The complete pseudo-code of our algorithm is given below:

Membership Form

/* Executes at each peer simultaneously */

Begin

Define ItemFile for attributes of peer database

Declare the claimed items set from ItemFile

Receive the claimed items set from neighbourhood ItemFile

Find intersection of personal item set with received claimed item set

If received claimed item is mapped to personal item, but not in personal set

Add claimed item in intersection set

End-If

Foreach item of intersection set

If item is not present in declared claimed items

Add item to declared claimed items

End-If

End-For

End

The differences between Khambatti and our membership formation algorithms are that we use peer databases instead of text files. We also use mapping tables to get claimed items in intersection sets.

Membership Discovery

/* Executes at each peer simultaneously */

Begin

Foreach claimed items

Compute link weight of each item of every peer

/* Calculation of link weight */

Link weight=0

Foreach peer P

Foreach neighbour P_1 of P

If neighbour P_1 **contains** claimed items

Link weight = Link weight + 1

End-If

Foreach neighbour of P_1

If neighbour **contains** claimed items

Link weight = Link weight + 1

End-If

End-For

End-For

End-For

/* Calculation of link weights of each peer is done by analyzing the claimed item sets that it receives from neighbouring peers. */

/* End of link weight calculation */

If link weight > T /* T= Threshold value */

Consider as a *community item*

End-If

End-For

/* [Number of Community Signature= 2^i-1; *'i'* is number of *community items*] */

For all subsets S of *community items*

Peer is a member of community with signature S

Set a **Vector** for community [v]

/* Keep community information in a vector array */

End-For

Store community peers ID in *CommunityBoard*

Refresh communities within a time interval

End

Our algorithm first calculates the link weight of all claimed attributes and compares the link weight with the threshold value. Khambatti's algorithm computes the link weight of all claimed attributes and then checks the link weights for the subset of claimed attributes with threshold value. Our algorithm does not check the subset of claimed attributes. For this reason, our algorithm reduces reiteration.

Community-based search using Query Translation

/* Translate the given query in searching desired community */

Begin

Set identifier P_Q for the peer creating the query

Mark the actual query as being the original query

Set meta-information [i] for the query item

/* meta-information is defined from condition part of SQL query or all attributes*/

Check Community peers /* Check the desired community in each peer */

If claimed items of vector for community [v] match with meta-information [i]

Ps is determined

Convert query Q to its corresponding T-Query Q_T=T /* Q is an SQL query */

/* Ps is the first searching community peer where processing of query starts. */

If mapping table is not empty

Computes T-query $Q'_T=\prod_{U'}(T \bowtie m)$ /* m= mapping table, $_{U'}$=attribute*/

Else

Computes $Q'_T = \prod_{U'} (T)$ from local peer database

End-If

Obtain the query Q' from Q'_T using disjunction for each row of attributes

/* Q' is a SQL query */

Copy the Query into its Community*Board* and Send result to P_Q

Neighbour peer of community **checks** Community*Board*

If claimed items of Neighbour peer match with meta-information

Translate the query and **Send** result to P_Q

Copy the Query into its own *CommunityBoard*

Else

Copy the Query into its own *CommunityBoard*

End-If

Else

Search for P_S

End-If

End

3.4 Discussion of Our Algorithm

Our algorithm has three parts: (1) forming communities using an attribute escalation method; (2) discovering communities comparing link weight with the threshold value; (3) translating the given query using T-query in the appropriate communities, which is called community-based search. The community-based search

procedure calls T-Query for translating the given query. The aforementioned community-based search can be done using an alternative approach.

The alternative approach for community-based query is simple. After discovering communities, every peer knows to which community it belongs. The given query checks its meta-information with the claimed attributes of the CommunityBoard. It searches for community peers and translates the given query for those peers in the querying peer. Then the querying peer sends the translated query directly to those community peers in order to get results.

The alternative approach is the following:

Pseudo-code:

Community-based search using Query Translation (Alternative approach)

*/ * Translate the given query in searching desired community */*

Begin

Identify querying peer ID

Mark the actual query as being the original query

Identify meta-information [i] for the query item

/* meta-information is defined from condition part of SQL query or all attributes*/

Check Community peers /* Check the desired community in each peer */

If claimed items of vector for community [v] match with meta-information [i]

Search for community peers, whose claimed items match with meta-information

Convert query Q to its corresponding T-Query Q_T=T /* Q is an SQL query */

If mapping table is not empty

Computes T-query $Q'_T = \prod_{U'}(T \bowtie m)$ /* m= mapping table, $_{U'}$=attribute*/

Else

Computes $Q'_T = \prod_{U'}(T)$ from local peer database

End-If

Translate the query Q' from Q'_T using disjunction for each row of attributes

/* Q' is an SQL query */

Send translated query into those peers for getting result in querying peer

End-If

End

The alternative approach is given for comparing with our community-based search algorithm which is mentioned in section 3.3. Our community-based search algorithm processes the given query in a community peer and proceeds for the next peer of a community. But, in the alternative approach, the algorithm does not process and proceeds for the next community peers. As each peer knows about his communities peers, the alternative approach can process the given query from the querying peer.

3.5 System Architecture of Community-based Search

Figure 3.8 represents a generic architecture for our community-based peer DBMS with its main functionalities. This architecture extends the Hyperion DBMS architecture with a community-based search component. All the modules of our community-based search architecture are described in the following.

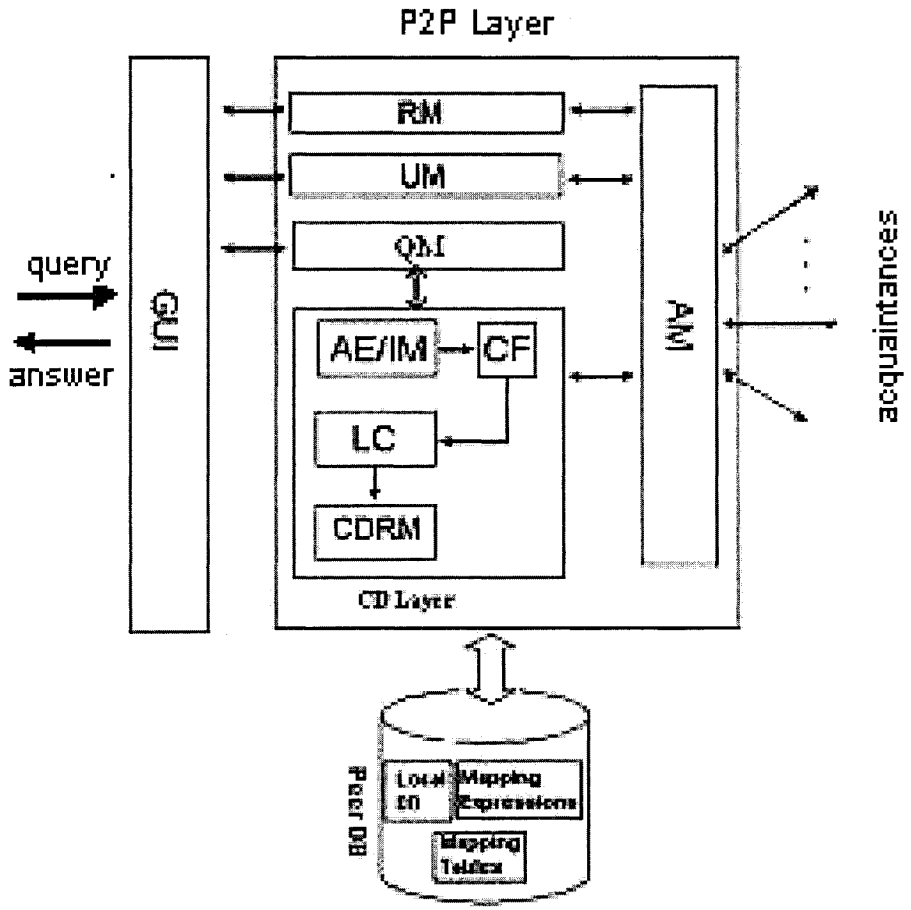

Figure 3.8: Architecture for a Community-based Search of a P2P System

This architecture consists of four main components: a interface (GUI), a P2P layer, a Community Discovery (CD) layer, and a DBMS. The GUI is the interface for posing queries (local or global) and specifying whether these are to be executed only locally or remotely.

The P2P layer has four sub-components: a Rule Manager (RM), a Update Manager (UM), a Query Manager (QM), and an Acquaintance Manager (AM). Through an AM, the P2P layer allows a peer database to establish or abolish an acquaintance (semi-) automatically at run time, thereby inducing a logical peer-to-peer network. The

AM uses mapping tables as a constraint on the data exchange between peer databases to automatically check the consistency of a set of mapping constraints and to infer new ones. The QM and the UM are responsible for queries and update propagation. They manage domain relations, coordination formulas, coordination rules, acquaintances, and interest groups. The RM enforces stabilization and consistency policies between peers. The Peer DB contains local databases, mapping tables, and mapping expressions that are used in data exchange with other peers.

The CD layer has four sub-components: an Attribute Escalator/Item Matcher (AE/IM), a Community Formator (CF), a Link weight Calculator (LC), and a Community Discovery Rule Manager (CDRM). The CD layer is connected with the QM and AM for propagating the given query into a desired community. The AE/IM escalates attributes/items by intersecting personal items with claimed items. The CF is responsible for forming communities. The LC calculates link weight of each of the claimed attributes. Finally, the CDRM discovers communities by comparing link weight with predetermined threshold values.

Chapter 4

Implementation and Experiments

4.1 Motivation

Efficient discovery of communities reduces the communication cost of community-based search. If communities are discovered in an optimized way, then query results among communities will be faster. The term "optimization" usually presumes the system retains the same functionality. However, a significant improvement in performance can often be achieved by solving only the actual problem and removing irrelevant functionality. In computing, optimization is the process of modifying a system to improve its efficiency. Our goal is to send the query after forming and discovering communities, or to discover communities 'on the fly' at the time of propagating the given query. For this reason, we are optimizing the community discovery procedure. The focus of this chapter is to discuss Khambatti's community discovery technique with a competent and optimized approach, and to elaborate shortly the idea of discovering communities 'on the fly'. This chapter also explains experimental results and discusses our prototype implementation.

4.2 Khambatti's Community Discovery Technique

Khambatti's community discovery algorithm is presented here to identify the difference between his community discovery algorithm and an optimized community discovery algorithm. His algorithm computes the link weight of all claimed attributes. It then checks the link weight of each claimed attribute for the subset of claimed attributes.

If all the attributes of each subset are greater than the pre-determined threshold value, then the peer is a community member of that subset. Khambatti's community discovery algorithm is as follows:

Membership Discovery (peer)

/* execute at each peer simultaneously */

Foreach claimed attributes

    Compute link weight

End-for

Foreach subset S of claimed attributes set

    Foreach attribute in S

        If link weight > T then

            Check next attribute

        Else

            Break-for /* try another subset */

    End-for

    If no break-for was executed

        Peer is member of community with signature S

End-for

4.3 Optimizing Khambatti's Community Discovery Technique

Our algorithm is the optimized form of Khambatti's community discovery algorithm. The optimized algorithm first calculates the link weight of all claimed attributes and compares the link weight with the threshold value. If the link weight is greater than the threshold value, then that attribute is considered as community items. In

this way, the algorithm finds the community items. After that, the algorithm finds the subset of these community items and all these subsets are the members of the community. The optimized algorithm is as follows:

Membership Discovery

/* Executes at each peer simultaneously */

Foreach claimed items

Compute link weight of each attribute in every peer

If link weight > T /* T= Threshold value */

Consider as a *community item*

End-If

End-For

/* [Number of Community Signature= 2^i-1; *'i'* is number of *community items*] */

For all subsets S of *community items*

Peer is a member of community with signature S

Set a **Vector** for community [v]

/* Keep community information in a vector array */

End-For

4.4 Benefit of Modified Technique

The modified community discovery technique reduces the comparing cost. As a result, it takes fewer iterations and processing is faster than Khambatti's algorithm. Suppose a peer has 10 claimed attributes. In this case, the peer will have 2^{10} (1024) subsets. If the link weight of 5 claimed attributes of that peer is less than the threshold value, then our modified algorithm takes 2^5 (32) iterations, instead of 2^{10} (1024)

iterations. But Khambatti's algorithm takes 2^{10} (1024) iterations, which is redundant. Our algorithm is faster than Khambatti's community discovery algorithm.

4.5 Discovering Communities 'on the fly'

4.5.1 Communities 'on the fly'

Communities can be discovered in a local machine (each peer), in a distributed way, or 'on the fly'. This section introduces how communities can be discovered 'on the fly'. Discovering community 'on the fly' means that when a peer poses a query then it propagates to the acquainted peers and discovers communities at runtime.

We assume that each peer calculates the link weight of each attribute of every acquainted peer or any initiator peer calculates the link weight and sends this calculated link weight file to the group where it belongs. Link weight is the key factor in discovering communities. Then we have to set a threshold value. The threshold value compared with the link weights generates the signature of the community. In this way, communities can be discovered.

If the discovered communities match with the received query criteria, then the query is translated to retrieve results from this peer.

4.5.2 Algorithm for Discovering Community 'on the fly'

The pseudo-code is as follows:

Begin

Calculate/Receive calculated link weight of each claimed attributes from initiator peer

Set a threshold value

If receive message from neighbor peer

Compare link weights of attributes mentioned in the message with Threshold value (T)

For each claimed attributes whose link weight > T

Peer belongs to community with Signature S

End-For

If Query Criteria match discovered community

Translate the given query for this peer

End-If

End-If

End

4.5.3 Discussion

The key factor for discovering community 'on the fly', which is computing the link weights, is the same as the regular discovery of communities in a local peer. The only difference is that communities are discovered in run time when a peer poses a query. The initiator peer sends the posed query to its acquaintees then, upon receipt, each acquainted peer starts discovering to which communities. After discovering communities at runtime, the algorithm translates the given query for communities peers. The advantage of this mechanism is that the peer does not need to store community information. As a result, every peer optimizes storage capacity. The main disadvantage is to discover communities in every execution time and the algorithm needs more memory space. The query completion time for this case is slower than our regular community-based search.

4.6 Implementation

We have implemented our community-based search algorithm which is mentioned in section 3.3. We have done experiments on our implemented algorithm. Section 4.7 shows our experimental results. We have not implemented our discovering community 'on the fly' algorithm which is mentioned in section 4.5.

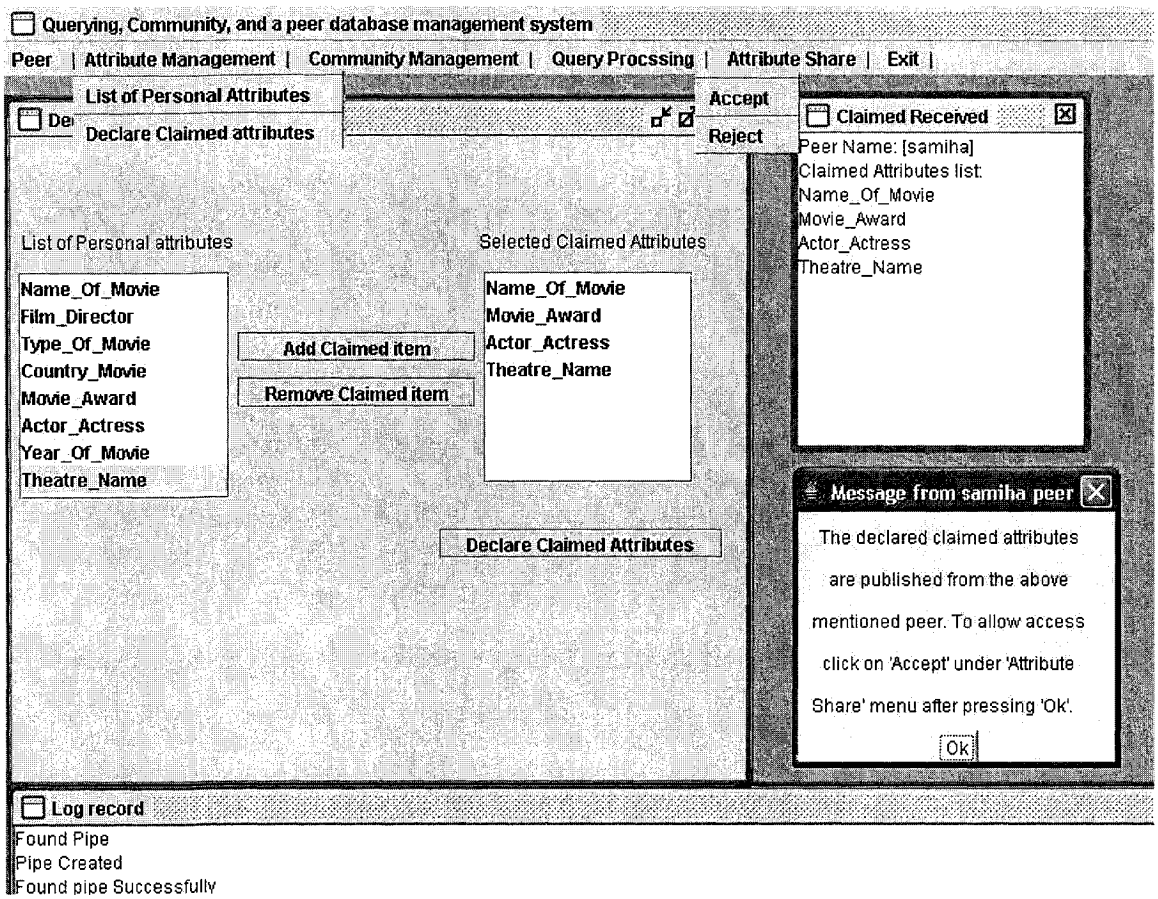

Figure 4.9: Screen-shot of the GUI of Declared Claimed Items

Our system is written in Java SDK 1.4.2 on top of JXTA, a P2P programming framework that alleviates the burden of writing huge source code. Its implementation consists of approximately 4000 lines of code. The prototype system runs on the Microsoft Windows environment. Figure 4.9 and 4.10 show screenshots of the query interface of our system.

Figure 4.9 shows the main interface for claiming attributes and managing claim acceptances and rejections. When a peer declares claimed attribute, then one message is generated for all acquainted peers. The acquainted peers then accept or reject their interest by pressing the Accept/Reject button. When a user chooses the "Accept'' button, the information is sent to the corresponding text files of all acquainted peers.

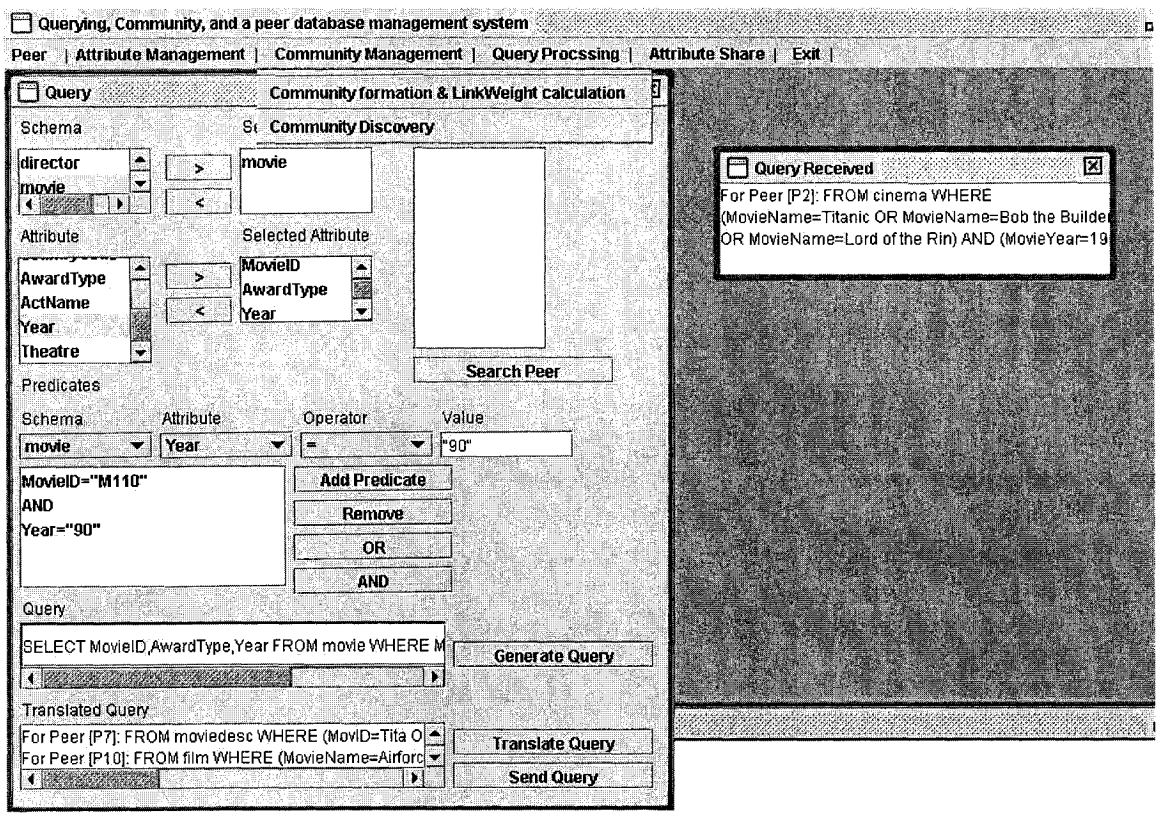

Figure 4.10: Screen-shot of Query Generation, Translation, and Community Discovery

Figure 4.10 shows the query interface which serves for generating queries, translating them, and discovering communities. It shows the community formation and link weight calculation and the community discovery menus as well. After generating the query, the peer translates it for community peers and sends the translated query to those

peers for getting query results. The user has to select schema, attributes, and predicates for generating queries. The given query uses mapping tables for translating the generated query.

4.7 Experiments

Our system uses MySQL 4.1.9 as our DBMS for the local DB layer. This system is built on top of JXTA. In our experiments, we found the performance of Java SDK 1.4.2 satisfactory. We provide a user interface for declaring claimed attributes/items, discovering communities, and posing a query to the peer database networks. The algorithm for community-based search is the combination of Khambatti's community formation and discovery technique, and the query translation method of Kementssietsidis using mapping table concepts. We develop an algorithm keeping the above ideas with supplementary modifications. We are mainly concerned with measuring the correctness and efficiency of discovering communities. To measure the efficiency and effectiveness of our algorithm, we did some experiments. We created a real network with 10 peers on 4 different machines using the JXTA package. We observed that the algorithms work well.

To measure the efficiency and effectiveness of our algorithm, we did some experiments. Because of the unreliability of JXTA pipes at the time of the experiments, and motivated by the desire to create a signification amount of peers, we decided to create two networks of 100 and 300 peers, respectively, all on one single machine. We experimented by setting threshold value at 40%, 50% and 70%. We get fewer communities with increasing threshold values. As we mentioned earlier (see section 3.1.1) threshold values are determined based on observations. Based on the fact that at

around 60%, sufficiently many communities are formed, we decided to set the link threshold to 60%. We collected data for evaluating the following:

- Community refreshment within different time intervals;
- Effect of query propagation for community changes;
- Percentage of community discovered for different sized peer networks.

4.7.1 Community Refreshment within Different Time Intervals

To do this measurement, we first had to find out the number of communities that were discovered given different time intervals. This information is then used to calculate the percentages of community changes. The X-axis represents time in minutes; the Y-axis denotes the corresponding percentage of community changes.

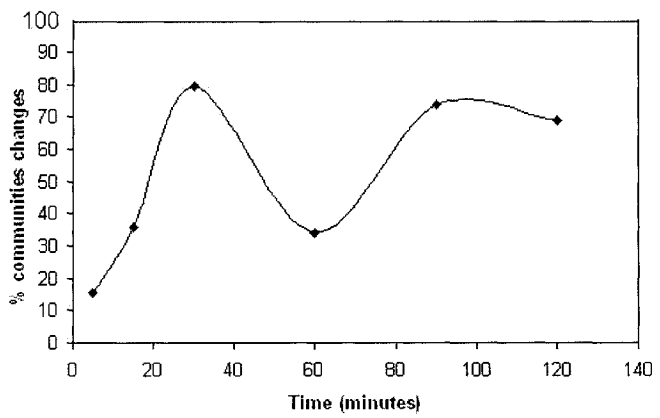

Figure 4.1: Community Refreshment by Changing Peers/Nodes.

Initially, we ran our simulation with 100 peer networks. For plotting percentage community changes, we gathered two sets of data: one is by changing peers (join or leave) from networks, the other is by varying attributes from peers within different time intervals. In our model, there are 4 text files and these are: network file, personal attribute file, claimed attribute file, and mapping table file. When a user accepts claimed

attributes, the corresponding claimed attributes and peer name are stored in a text file named claimed.txt, similarly when peers join the network, the peer name and link are stored in a text file named network.txt. In our experiment, joining or leaving from networks means to add or delete peer name from the network.txt text file. We manually change entries in network.txt file. In our networks, each peer contains a maximum of 12 personal attributes, and a peer was explicitly placed into only one group.

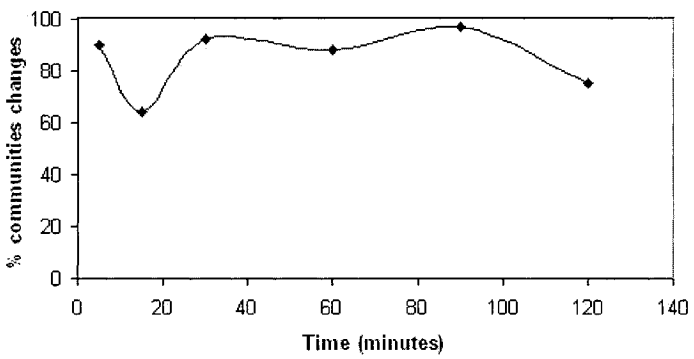

Figure 4.2: Community Refreshment by Changing Attributes/Items.

The graphs shown in Figure 4.1 and 4.2 are the results of our community refreshment within different time intervals. The curves of Figure 4.1 and 4.2 are in different shapes. The percentage of communities change depends on how many peers change their status after community refreshment. The large time interval for community refreshment is more efficient because the experiment shows that a few users change their position (leave or join) over a small time interval. For attribute changes, the percentage of community changes is high and always almost over 80%, but for peer/node changes, the percentage of community changes is not so high and varies often. Our observation determines that a peer changes its attributes more frequently than to join or leave networks.

4.7.2 Effect of Query Propagation for Community Changes

For this evaluation, we send a query; change the percentage of communities, and record query completion time. In our 100 peer network, we placed approximately 300 mapping tables.

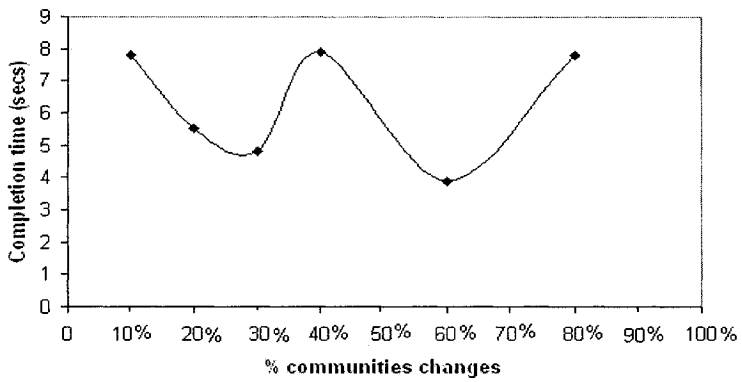

Figure 4.3: Effect of Community Changes on the Completion Time a Given Query: SELECT MovieID,Year FROM movie WHERE MovieID="M110" OR Year="00"

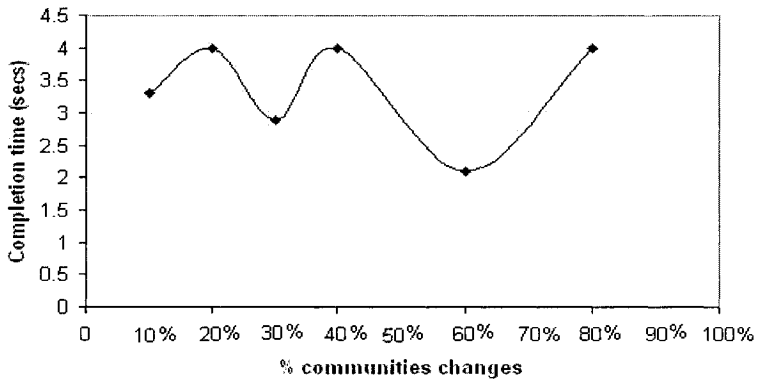

Figure 4.4: Effect of Community Changes on the Completion Time a Given Query: SELECT MovieID,Year FROM movie WHERE MovieID="M110" AND Year="00"

The graphs shown in Figures 4.3 and 4.4 are the results of our experiment. The X-axis indicates percentage of community changes and Y-axis represents query completion time in seconds. We send different queries by varying percentage communities and plot separately.

The curve of Figure 4.3 goes down, then up, and then down whereas Figure 4.4 goes up, then down, and then up. In the first few communities change of Figure 4.3, the response of peers decreases for disjunctive query. Again response of peers increases. The response of peers depends on attribute matching of a given query with its claimed attributes. Similarly, in the first communities change of Figure 4.4, the response of peers increases for conjunctive query. Again response of peers decreases. Due to the increase and decrease of peers response, the curves of Figure 4.3 and 4.4 go up and down. We have done our experiments on fixed sizes of network and our declared claimed attributes, but it may vary at a time in real P2P environment.

The completion time for a specific query does not depend on the size of the query. Query completion time depends on how many peers reply after changing communities. We did not send complex queries since that was not our goal. Our target was to observe the query completion time for given percentages of community changes. We sent two SQL queries where the condition part of the first one was a conjunction (AND), and the second one was a disjunction (OR). Experiments indicate that simple disjunctive queries take approximately double the time that conjunctive queries do.

4.7.3 Percentage of Communities Discovered for Different Sizes of Peer Networks

We ran our algorithm in different sized peer networks and recorded the number of communities discovered. We started to run our prototype with 50 peers, increasing 50 peers every time, and taking the measurement up to 200 peers.

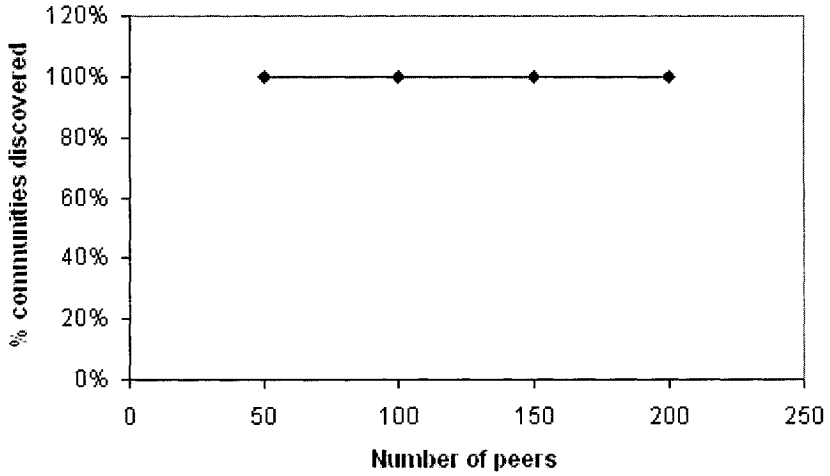

Figure 4.5: Effect of Communities Discovered for Different Sizes of Networks

Figure 4.5 depicts percentage of community discovered for different sized networks. The X-axis represents number of peers/nodes and the Y-axis represents percentage of communities discovered. We observe that the percentage of communities discovered remains the same all the time. The number of communities for each peer should be (2^i-1), where 'i' is the number of claimed attributes whose link weight is greater than predetermined threshold value. We verified that the number of discovered communities is correct. We also verified the calculated link weight for validating our discovered communities. The number of discovered communities remains constant for different sized networks and discovers 100% of communities.

4.7.4 Complexity of Our Algorithm

A community-based search algorithm is our main objective. This algorithm has three major parts: community formation, community discovery, and query translation using T-query. In order to measure the overall complexity, we first discuss the

complexity of related procedures and finally we determine the complexity of our community-based search algorithm.

The development of fast and efficient algorithms is very important for computers. Generally, to estimate the complexity of an algorithm, we need to determine how much time and space is required to implement it. This requires determining the number of steps that a program will take to solve a problem. To count steps, it is important to consider the order of growth of a problem. The complexity of an algorithm can be determined theoretically and empirically. We measure the complexity of our algorithm empirically, concentrating on how long it takes to perform our desired experimental operation.

A community can be discovered in each peer, or in a distributed way, or 'on the fly'. We implemented the community discovery technique in each peer simultaneously. The running time depends on the programming language that is used for implementation and the machine that the program is run on. As our main input elements are attribute and node/peer, we input different sizes of attributes and nodes/peers separately or collectively. We then observe the results of community formation and link weight calculation time, community discovery time, community search of a given query & query translation time for determining the complexity of the algorithm.

We analyze the complexity of our algorithm with the following experiments. We started using 12 attributes and increased up to 50. The required time for the same operation remains the same for nodes or attributes change. For simplicity, only the operations for node changes are shown.

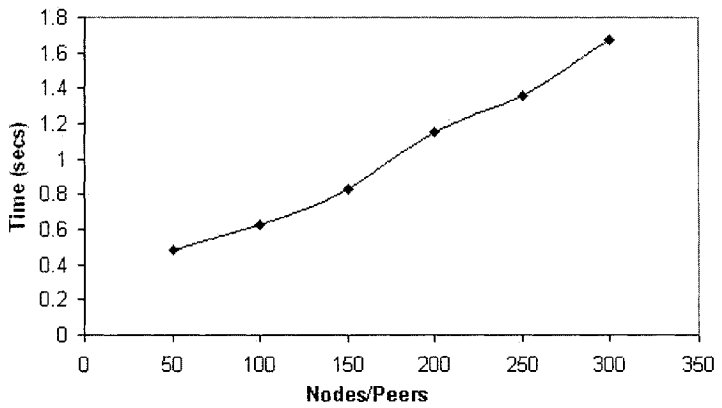

Figure 4.6: Community Formation and Link Weight Calculation

Figure 4.6 illustrates community formation and link weight calculation time in seconds for different sized networks. We started with 50 nodes and increase by 50 nodes every time. We gather data 6 times until 300 nodes to observe community formation and link weight calculation time. From the above graph, we see that community formation and link weight calculation time increases gradually in terms of node increases. This algorithm does not take long to perform the task. We see that time increases slightly (milliseconds) for node changes.

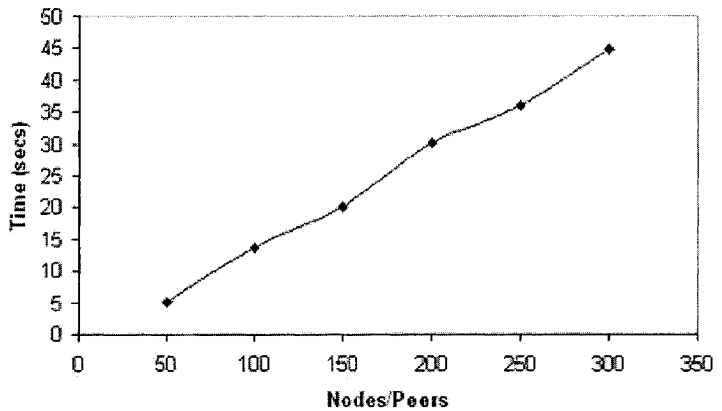

Figure 4.7: Community Discovery Times

Figure 4.7 depicts the community discovery time in terms of different sized networks. The same number of nodes and iterations we used for community formation

and link weight calculation experiment are used here. We set the link threshold to 60%. From the above graph we see time increases progressively for node changes. Community discovery method takes longer than link weight calculation time. The increasing time for community discovery is much more than to form community for the same network changes. The reason is that for discovering communities the algorithm compares link weight with threshold values every time whereas for community formation and link weight calculation it only calculates values, there is no comparison.

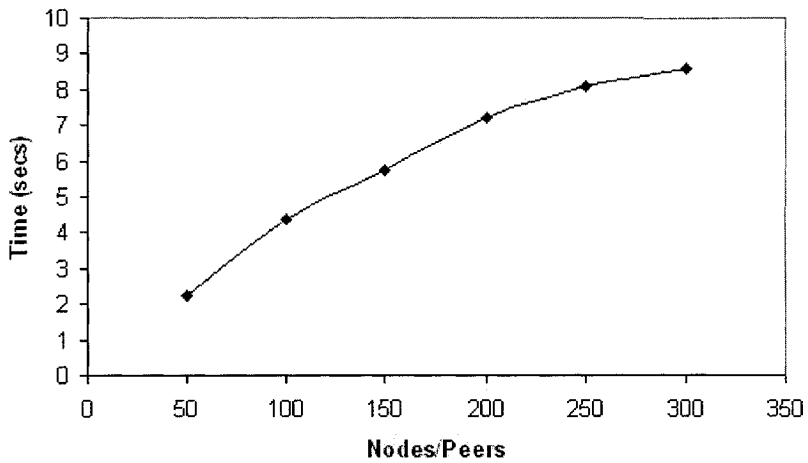

Figure 4.8: Overall Query Processing Time for a Typical Simple Query

Figure 4.8 shows the community search of a given query and query translation time for different sized networks. This is our main module. This module is the community-based search module. Figure 4.8 shows that time increases almost proportionally in terms of network changes. The increasing time for network changes is in between link weight calculation and community discovery module. The required increasing time is in seconds but is not high.

In every case, the function for time complexity is different, but time increases gradually in terms of node increases. The coefficient slightly varies in all cases. The

community discovery module takes longer in comparison with other module (e.g. link weight calculation, query translation, etc.). But we do not need to discover communities every time. We need to refresh communities in a certain interval for accurate performance. However, the community-based search module is not slower. Based on the experiments, we can observe that our algorithm takes gradual time increments in terms of node increments.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

P2P systems offer several advantages in simplicity of use, robustness, and scalability. In this thesis we address the issue of forming and discovering communities, and pose a query to get response from appropriate communities. We analyze some techniques for discovering communities in a text files and then present an algorithm.

Our prototype consists of five components: an attribute declaration mechanism, link weight calculation module, community discovery mechanism, a query generator tool, and a query translation scheme for community peers. The attribute declaration mechanism declares claimed attributes and propagates to the group. The acquainted peer accepts/rejects its interest regarding these claimed attributes. If it accepts, a message is sent to all peers each of which stores this information in its claimed text file. The link weight calculation module calculates link weight and stores this result in a text file. The Community discovery mechanism discovers communities by taking information from claimed and link weight text files. A query generator tool generates the query based on predicate selection, and a query translation scheme translates the given query for every community peer through mapping tables.

Our experimental results indicate that community discovering remains accurate in different network sizes. Experiments show that we need more memory swap space if we increase the number of claimed attributes. In that case, the community sorting period also

increases. Experiments also indicate that the threshold value has a strong impact on discovering communities. So, we have to determine the threshold value in a careful way, otherwise we will not reach our expected communities.

Users can identify their peer names as numeric or alpha numeric also. We develop our prototype on top of JXTA, which supports P2P platform environment. An approach for discovering community 'on the fly' is introduced.

5.2 Future Work

For future work we plan to implement the algorithm for discovering communities 'on the fly'. We plan to collect real data and use them to experiment on our prototype. Currently, our system accepts only simple queries (e.g. AND, OR), but complex queries could be incorporated in our algorithm as well.

We further plan to discover communities in a distributed way. Automatic mapping table generation is necessary for query translation purposes, which is not our objective, but in some cases it is required to discover an accurate community. There is also potential to extend our prototype so that it can run in a real environment.

Based on observations, it is recommended that if there are a large number of claimed attributes in the network, then any single peer should work on a portion of claimed attributes for discovering communities and other peers should work on some portion of attributes; similarly the next peer should work on some of the attributes to discover communities until the end of claimed attributes. After discovering communities for all claimed attributes, then all discovered communities in a file should be merged and sent to all peers in the community.

BIBLIOGRAPHY

[1] Aberer, K. and Hauswirth, M., "Peer-to-Peer Information Systems: Concepts and Models, State-of-the-art, and Future Systems," Tutorial at ICDE 2002 Advanced Technology Seminar, San Jose, CA, 2002.

[2] Arenas, M., Kantere, V., Kementsietsidis, A., Kiringa, I., Miller, R.J., and Mylopoulos, J., "The Hyperion Project: From Data Integration to Data Coordination," *In SIGMOD Record, Special Issue on Peer-to-Peer Data Management*, 32(3):53-58, 2003.

[3] Bernstein, P.A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., and Zaihrayeu, I., "Data Management for Peer-to-Peer Computing: A Vision," *Proceedings of the 5th International Workshop on the Web and Databases (WebDB 2002)*, Madison, Wisconsin, 2002.

[4] Cheung, K., Kantere, V., Kementsietsidis, A., Kiringa, I., Miller, R.J., Mylopoulos, J., and Wang, J., "Hyperion: A Network of Peer Database Management Systems Using Data Coordination," Submitted for publication (*SIGMOD 2004*).

[5] Dabek, F., Bruunskill, E., Kaashoek, M.F., Karger, D., Morris, R., Stoica, I., and Balakrishnan, H., "Building Peer-to-Peer Systems with Chord, A Distributed Lookup Service," *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, May 2001.

[6] Doucet, A. and Lumineau, N., "A Collaborative Approach for Query Propagation in Peer-to-Peer Systems," University of Paris 6, France, SWDB, 2003, pp. 251-257.

[7] Flake, G.W., Lawrence, S., and Giles, C.L., "Efficient Identification of Web Communities," *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD-2000),* 2000.

[8] Flake, G.W., Lawrence, S., Giles, C.L., and Coetzee, F.M., "Self-organization and Identification of Web Communities," IEEE computer, 35(3), 2002, pp. 66-71.

[9] Gibson, D., Kleinberg, J., and Raghavan, P., "Inferring Web Communities from Link Topology," Dept. of Computer Science. UC Berkeley Berkeley, USA, June 1999.

[10] Giunchiglia, F. and Zaihrayeu, I., "Making Peer Databases Interact - A Vision for An Architecture Supporting Data Coordination," *Proceedings of the Conference on Information Agents (CIA 2002)*, Madrid, September 2002.

[11] Google. http://www.google.com/

[12] Gnasa, M., Alda, S., Grigull, J., and Cremers, A.B., "Towards Virtual Knowledge Communities in Peer-to-Peer Networks," *Proceeding of the SIGIR Workshop on Distributed Information Retrieval. Workshop at the 26th International ACM SIGIR Conference*, LNCS 2974, Springer, Toronto, Canada, August 2003.

[13] Gribble, S., Halevy, A., Ives, Z., Rodrig, M., and Suciu, D., "What Can Databases Do for Peer-to-Peer?," *Proceedings of the Fourth International Workshop on the Web and Databases*, Santa Barbara, CA, USA, 2001.

[14] Hoschek, W., "A Unified Peer-to-Peer Database Protocol," Data Grid TechReport DataGrid-02-TED- 0407, April 2002.

[15] Jxta project, (Online) Available at: http://www.jxta.org.

[16] Kalogeraki, V., Gunopulos, D., and ZeinalipourYazti, D., "A Local Search Mechanism for P2P Networks," *CIKM'02,* November 4–9, McLean, Virginia, USA, 2002.

[17] Kantere, V., Kiringa, I., Mylopoulos, J., Kementsietsidis, A., and Arenas, M., "Coordinating Peer Databases Using ECA Rules," *International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, September 2003.

[18] Kantere, V., Mylopoulos, J., and Kiringa, I., "A Distributed Rule Mechanism for Multi-database Systems," *International Conference on Cooperative Information Systems (CoopIS)*, November 2003.

[19] Kaykova, O., Kononenko, O., Terziyan, V., and Zharko, A., "Community Formation Scenarios in Peer-to-Peer Web Service Environments," Department of Mathematical Information technology, University of Jyvaskyla, Finland, February 2004.

[20] Kementsietsidis, A., Arenas, M., and Miller, R.J., "Managing Data Mappings in the Hyperion Project," *Proceedings of the International Conference on Data Engineering (ICDE) 2003*, pp. 732-734, March 2003.

[21] Kementsietsidis, A., Arenas, M., and Miller, R.J., "Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues," *Proceedings of the ACM SIGMOD International Conference on Management of Data,* pp. 325-336, June 2003.

[22] Kementsietsidis, A. and Arenas, M., "Data Sharing through Query Translation in Autonomous Sources," *Proceedings of the 30th VLDB conference*, Toronto, Canada, 2004.

[23] Khambatti, M., Ryu, K.D., and Dasgupta, P., "Efficient Discovery of Implicitly Formed Peer-to-Peer Communities," *International Journal of Parallel and Distributed Systems and Networks*, 5(4), 2002, pp. 155-164.

[24] Khambatti, M., Ryu, K.D., and Dasgupta, P., "Peer-to-peer Communities: Formation and Discovery," *14th IASTED International Conference on Parallel and Distributed Computing Systems (PDCS 2002)*, Cambridge, MA, November 2002.

[25] Khambatti, M., Ryu, K.D., and Dasgupta, P., "Push-Pull Gossiping for Information Sharing in Peer-to-Peer Communities," *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pp. 1393-1399, Las Vegas, Nevada, June 2003.

[26] Khambatti, M., Ryu, K.D., and Dasgupta, P., "Structuring Peer-to-Peer Networks Using Interest-Based Communities," *International Workshop on Databases, Information Systems and Peer-to-Peer Computing (P2PDBIS)*, Humboldt University, Berlin, Germany, September 2003.

[27] Khambatti, M., "Peer-to-Peer communities: Architecture, Information and Trust Management," Ph.D. Thesis, Arizona State University, December 2003.

[28] Lenzerini, M., "Data Integration: A theoretical Perspective," *Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS 2002, Madison, Winsconsin, USA, June 2002.

[29] Litwin, W., Mark, L., and Roussopoulos, N., "Interoperability of Multiple Autonomous Databases," ACM Computing Surveys (CSUR), v.22 n.3, pp.267-293, ACM Press, New York, September 1990.

[30] Lumineau, N. and Doucet, A., "Sharing Communities Experiences for Query Propagation in Peer-to-Peer Systems," University of Paris 6, PADOUE project (http://www-poleia.lip6.fr/padoue) financed by ACI GRID: http://wwwsop.inria.fr/aci/grid/public.

[31] Madhavan, J., Bernstein, P.A., Domingos, P., and Halevy, A.Y., "Representing and Reasoning about Mappings between Domain Models," *Proceedings of the Eighteenth National Conference on Artificial Intelligence,* Edmonton, Alberta, Canada, 2002, pp. 80–86.

[32] Maymounkov, P. and Mazieres, D., "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," IPTPS 2002, Cambridge, MA, USA, March 2002.

[33] Piazza peer data management system (pdms), (Online) Available at: http://data.cs.washington.edu/p2p/piazza

[34] Serafini, L., Giunchiglia, F., Mylopoulos, J., and Bernstein, P.A., "Local Relational Model: A Logical Formalization of Database Coordination," Technical Report DIT-03-002, Informatica e Telecomunicazioni, University of Trento, June 2003.

[35] Sheth, A.P. and Larson, J.A., "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases," ACM Computing Surveys (CSUR), v.22 n.3, pp.183-236, ACM Press, New York, September 1990.

[36] Siong Ng, W., Ooi, B.C., Tan, K.L., and Zhou, A., "PeerDB: A P2P-based System for Distributed Data Sharing," *Proceedings of the 19th International Conference on Data Engineering*, Bangalore, India, 2003, pp. 633-644.

[37] Sripanidkulchai, K., Maggs, B., and Zhang, H., "Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems," *Proceedings of INFOCOM 2003*. A *poster* for ACM SIGCOMM Computer Communication Review, 32(1), January 2002.

[38] Vassileva, J., "Supporting Peer-to-Peer User Communities," *CoopIS/DOA/ODBASE* 2002, LNCS 2519, 2002, pp. 230-247.

[39] Yang, B. and Garcia-Molina, H., "Efficient Search in Peer-to-Peer Networks," *International Conference On Distributed Computing System*, Vienna, Austria, 2002.

[40] Zaihrayeu, I., "Query Answering in Peer-to-Peer Database Networks," *Technical Report # DIT-03-012*, University of Trento, March 2003.

APPENDIX A

SAMPLE OUTPUTS

1. Link weight of each attribute for every peer

(For simplicity & to save space, 10 out of 100 peer link weights are shown)

PeerName	ClaimedAttribute	ClaimedDBField	LinkWeight
0	Type_Of_Movie	TypeID	64
0	Name_Of_Movie	MovieID	61
0	Country_Movie	CountryCode	59
0	Cost_Of_Movie	MovCost	58
0	Main_Singer	Singer	43
0	Film_Director	DirName	59
0	Theatre_Name	Theatre	64
0	Year_Of_Movie	Year	68
0	Movie_Award	AwardType	60
0	Best_Selling	Selling	58
0	Movie_Producer	Producer	55
1	Theatre_Name	Theatre	67
1	Year_Of_Movie	Year	68
1	Type_Of_Movie	TypeID	64
1	Name_Of_Movie	MovieID	61
1	Country_Movie	CountryCode	57
1	Cost_Of_Movie	MovCost	58
1	Main_Singer	Singer	44
1	Film_Director	DirName	58
1	Best_Selling	Selling	59
1	Movie_Producer	Producer	57
2	Name_Of_Movie	MovieID	43
2	Type_Of_Movie	TypeID	47
2	Main_Singer	Singer	35
2	Film_Director	DirName	44
2	Movie_Award	AwardType	44
2	Best_Selling	Selling	40
2	Movie_Producer	Producer	41
3	Name_Of_Movie	MovieID	33
3	Year_Of_Movie	Year	38
3	Main_Singer	Singer	25
3	Film_Director	DirName	33
3	Theatre_Name	Theatre	35
4	Best_Selling	Selling	58
4	Movie_Award	AwardType	63
4	Type_Of_Movie	TypeID	67
4	Name_Of_Movie	MovieID	64
4	Country_Movie	CountryCode	60
4	Actor_Actress	ActName	51
5	Best_Selling	Selling	26
5	Name_Of_Movie	MovieID	30
5	Theatre_Name	Theatre	34
5	Actor_Actress	ActName	23

5	Movie_Award	AwardType	30
5	Year_Of_Movie	Year	35
5	Film_Director	DirName	31
5	Main_Singer	Singer	23
5	Movie_Producer	Producer	30
6	Actor_Actress	ActName	33
6	Name_Of_Movie	MovieID	42
6	Film_Director	DirName	44
6	Year_Of_Movie	Year	48
6	Movie_Producer	Producer	39
6	Theatre_Name	Theatre	47
6	Cost_Of_Movie	MovCost	40
6	Movie_Award	AwardType	44
6	Country_Movie	CountryCode	42
6	Type_Of_Movie	TypeID	47
6	Best_Selling	Selling	39
7	Actor_Actress	ActName	39
7	Theatre_Name	Theatre	54
7	Year_Of_Movie	Year	54
7	Type_Of_Movie	TypeID	52
7	Name_Of_Movie	MovieID	49
7	Country_Movie	CountryCode	46
7	Cost_Of_Movie	MovCost	44
7	Film_Director	DirName	45
7	Best_Selling	Selling	44
7	Movie_Producer	Producer	46
7	Movie_Award	AwardType	48
8	Theatre_Name	Theatre	43
8	Type_Of_Movie	TypeID	45
8	Name_Of_Movie	MovieID	45
8	Country_Movie	CountryCode	43
8	Year_Of_Movie	Year	45
8	Movie_Award	AwardType	43
8	Best_Selling	Selling	43
8	Movie_Producer	Producer	39
9	Type_Of_Movie	TypeID	28
9	Cost_Of_Movie	MovCost	26
9	Movie_Producer	Producer	23
9	Best_Selling	Selling	27
9	Actor_Actress	ActName	23
9	Main_Singer	Singer	14
9	Theatre_Name	Theatre	26
9	Name_Of_Movie	MovieID	28
9	Year_Of_Movie	Year	28
10	Name_Of_Movie	MovieID	42
10	Country_Movie	CountryCode	42
10	Year_Of_Movie	Year	48
10	Movie_Award	AwardType	44
10	Film_Director	DirName	44
10	Type_Of_Movie	TypeID	47
10	Actor_Actress	ActName	33
10	Cost_Of_Movie	MovCost	40
10	Main_Singer	Singer	35
10	Theatre_Name	Theatre	47
10	Movie_Producer	Producer	39

2. Signature of community with members (peers)

(For simplicity & to save space, not all communities are shown)

Signature of Community with member peers:

[MovieID = Name_Of_Movie, DirName = Film_Director, TypeID = Type_Of_Movie,
CountryCode = Country_Movie, AwardType = Movie_Award, ActName = Actor_Actress,
Year = Year_Of_Movie, Theatre = Theatre_Name, Selling = Best_Selling, MovCost = Cost_Of_Movie,
Singer = Main_Singer, Producer = Movie_Producer]

Community Signature Peer Name

[TypeID] ----------> {0,1,2,4,6,7,8,9,10,12,15,17,19,21,22,23,24,25,26,28,29,31,32,33,34,35,38,39,41,42,
43,44,45,46,50,51,52,53,54,60,64,73,75,76,77,86,89,91,93,96,98,99}
[MovieID] ----------> {0,1,2,3,4,5,6,7,8,9,10,12,14,15,16,17,18,19,20,21,22,27,28,29,30,31,32,33,35,38,
39,40,42,44,47,49,50,52,53,54,55,57,60,61,62,65,70,71,75,77,81,82,85,90,91,93,96,97,99}
[CountryCode] ----------> {0,1,4,6,7,8,10,11,12,14,17,18,21,22,25,26,28,29,30,31,32,33,34,35,36,37,38,
39,42,45,46,49,50,52,53,54,57,60,64,71,79,89,91,96,98,99}
[MovCost] ----------> {0,1,6,7,9,14,18,19,21,22,24,28,29,31,32,33,34,35,37,38,39,40,42,44,45,49,52,53,
54,57,60,61,64,71,73,76,80,83,86,90,91,92,93,94,96,97,98}
[Singer] ----------> {0,1,2,3,5,9,14,15,18,19,21,24,25,30,31,34,35,37,38,39,40,50,52,54,71,75,77,89,91,
94,98}
[DirName] ----------> {0,1,2,3,5,6,7,10,11,12,14,15,16,17,18,19,21,22,24,28,31,32,33,34,36,37,38,39,42,
44,46,52,54,57,60,64,65,75,77,89,90,91,93,94,96,98,99}
[Theatre] ----------> {0,1,3,5,6,7,8,9,12,15,17,18,21,22,23,24,25,28,29,31,34,35,37,38,39,40,42,43,44,45,
46,47,49,50,52,54,57,61,64,71,72,73,75,76,77,79,89,91,94,98,99}
[Year] ----------> {0,1,3,5,6,7,8,9,10,11,12,14,15,17,18,20,21,22,24,25,27,28,29,31,34,35,38,39,42,44,
45,47,49,52,53,55,56,57,61,62,63,64,65,66,69,71,75,76,77,79,80,81,82,83,85,86,89,90,93,94,96,98,99}
[AwardType] ----------> {0,2,4,5,6,7,8,10,12,15,17,18,21,22,31,33,34,35,37,38,39,40,42,44,45,46,47,48,
49,52,53,54,55,57,60,61,64,65,71,75,76,77,81,86,94,96,97,98,99}
[Selling] ----------> {0,1,2,4,5,6,7,8,9,12,14,15,19,22,23,24,25,28,33,34,37,43,44,50,52,54,60,76,77,80,83,
89,91,93,98,99}
[Producer] ----------> {0,1,2,5,6,7,8,9,10,12,14,17,19,21,22,24,25,28,29,31,32,33,34,35,39,42,44,45,47,50,
53,54,57, 60,71,72,75,76,77,91,93,96,97,99}
[TypeID, MovieID] ----------> {0,1,4,7,8,9,12,15,17,19,21,22,28,29,32,33,35,38,42,44,50,52,54,60,75,77,
93,99}
[TypeID, CountryCode] ----------> {0,1,4,7,8,10,12,17,21,22,25,26,28,29,34,35,38,39,45,46,50,53,89,96,
98,99}
[TypeID, MovCost] ----------> {0,1,7,9,19,21,22,24,28,29,34,35,38,39,42,44,45,52,54,60,73,76,86,96}
[TypeID, Singer] ----------> {0,1,2,9,15,21,24,34,35,38,50,52,54,75,89,98}
[TypeID, DirName] ----------> {0,1,2,7,12,15,17,19,21,22,24,28,34,38,42,46,60,77,91,98,99}
[TypeID, Theatre] ----------> {0,7,9,12,15,17,22,23,24,25,28,34,38,39,42,44,46,52,54,73,75,76,89,91}
[TypeID, Year] ----------> {0,7,8,9,10,12,15,17,22,24,25,28,34,38,39,42,44,52,75,76,77,89,93,98}
[TypeID, AwardType] ----------> {0,2,7,8,12,17,21,22,34,35,38,44,45,46,52,60,75,76,98,99}
[TypeID, Selling] ----------> {0,1,2,6,7,8,9,12,15,19,22,24,25,28,33,34,44,50,52,54,60,76,89,91,93}
[TypeID, Producer] ----------> {0,1,2,7,8,9,10,12,17,19,22,24,25,28,29,34,42,44,45,54,60,75,93,99}
[MovieID, CountryCode] ----------> {0,1,4,6,7,8,10,12,14,17,18,21,22,28,29,30,31,35,38,39,50,53,71,91,
96,99}
[MovieID, MovCost] ----------> {0,1,6,7,18,19,21,22,28,29,31,35,39,40,44,49,52,53,54,60,61,71,90,91,
96,97}
[MovieID, Singer] ----------> {0,1,2,3,5,9,14,15,21,30,31,35,40,50,52,54,71,75,91}
[MovieID, DirName] ----------> {0,1,2,3,5,6,7,10,12,15,16,17,18,19,21,22,28,42,57,60,77,90,91,99}

[MovieID, Theatre] ----------> {0,3,5,6,7,9,12,17,18,22,31,38,39,40,42,44,49,52,54,57,75,91}
[MovieID, Year] ----------> {0,3,5,6,7,8,9,10,12,14,17,18,20,22,31,39,42,44,47,49,52,53,55,57,71,75,77,
85,90,93}
[MovieID, AwardType] ----------> {0,2,5,6,7,8,10,12,17,18,21,22,31,35,38,40,44,52,53,55,60,61,71,75,
81,96,97,99}
[MovieID, Selling] ----------> {0,1,2,6,7,8,12,15,19,22,28,44,50,52,60,91,93}
[MovieID, Producer] ----------> {0,1,2,5,6,7,8,10,17,19,22,28,29,42,44,47,53,54,57,60,71,75,91,93,99}
[CountryCode, MovCost] ----------> {0,1,7,18,21,22,28,29,31,34,35,37,39,42,45,49,52,54,60,71,91,96}
[CountryCode, Singer] ----------> {0,1,14,21,30,31,34,35,37,50,52,54,71,98}
[CountryCode, DirName] ----------> {0,1,7,17,18,21,22,28,34,36,37,42,46,54,57,60,91,99}
[CountryCode, Theatre] ----------> {0,7,17,18,22,25,34,38,39,42,46,49,52,54,57,91}
[CountryCode, Year] ----------> {0,7,8,10,11,17,18,22,25,34,39,42,49,52,57,71}
[CountryCode, AwardType] ----------> {0,7,8,17,18,21,22,31,34,35,37,38,45,46,52,60,64,71,99}
[CountryCode, Selling] ----------> {0,1,6,7,8,22,28,33,34,37,50,52,54,60,91}
[CountryCode, Producer] ----------> {0,1,7,8,10,17,22,25,28,29,32,34,42,45,54,57,60,71,99}
[MovCost, Singer] ----------> {0,1,9,14,21,24,31,34,35,40,52,54,71,94,98}
[MovCost, DirName] ----------> {0,1,7,18,19,21,22,24,28,32,37,38,42,57,60,64,91,94,98}
[MovCost, Theatre] ----------> {0,7,9,18,22,24,34,38,42,44,49,52,54,57,64,73,76,91,94}
[MovCost, Year] ----------> {0,7,9,14,18,22,24,38,42,44,49,52,53,57,64,71,76,80,83,93,94,98}
[MovCost, AwardType] ----------> {0,6,7,18,21,22,31,34,35,37,38,44,45,52,57,60,61,64,71,76,94,97,98}
[MovCost, Selling] ----------> {0,1,6,7,9,19,22,24,28,33,34,37,44,52,60,76,91,93}
[MovCost, Producer] ----------> {0,1,7,9,19,22,24,28,29,32,34,42,44,45,53,54,57,60,71,93}
[Singer, DirName] ----------> {0,1,2,3,15,18,19,21,24,37,38,39,77,91}
[Singer, Theatre] ----------> {0,3,9,18,24,25,34,38,39,52,75,77,91}
[Singer, Year] ----------> {0,9,18,24,25,38,39,52,71,75,77,89,94}
[Singer, AwardType] ----------> {0,2,18,21,34,35,37,38,39,52,71,75,94}
[Singer, Selling] ----------> {0,1,2,15,19,24,25,34,37,50,52,77,89,91}
[Singer, Producer] ----------> {0,1,2,5,19,24,25,34,39,71,75,77,91}
[DirName, Theatre] ----------> {0,3,6,7,17,18,22,24,31,34,38,39,42,44,46,52,54,75,89,91}
[DirName, Year] ----------> {0,6,7,10,11,12,14,17,18,22,24,31,38,39,42,44,52,65,75,77,89,90,
93,94,96,98}
[DirName, AwardType] ----------> {0,2,6,7,12,17,18,21,22,31,33,34,37,38,44,46,52,60,64,75,94,96,99}
[DirName, Selling] ----------> {0,1,2,6,7,14,15,19,22,24,28,33,34,37,44,52,54,60,89,91,93}
[DirName, Producer] ----------> {0,1,2,5,6,7,10,14,17,19,22,24,28,32,33,34,44,54,57,60,75,93,96,99}
[Theatre, Year] ----------> {0,1,5,7,8,9,12,17,18,21,22,24,25,28,35,39,42,44,45,47,49,52,57,61,64,71,75,
76,77,79,89,94,98,99}
[Theatre, AwardType] ----------> {0,5,6,7,8,12,17,18,21,22,31,34,35,37,38,44,45,46,52,61,64,71,75,76,
94,98,99}
[Theatre, Selling] ----------> {0,1,6,7,8,15,22,24,28,34,37,43,44,50,52,76,89,91,98}
[Theatre, Producer] ----------> {0,1,5,7,8,17,22,24,25,28,29,34,44,45,47,50,57,71,72,75,77,99}
[Year, AwardType] ----------> {0,6,7,8,12,17,18,21,22,31,34,35,38,44,45,52,61,64,71,75,76,81,94,96,99}
[Year, Selling] ----------> {0,1,6,7,8,15,22,24,28,34,44,52,76,89,93}
[Year, Producer] ----------> {0,1,5,6,7,8,10,17,22,24,25,28,29,34,44,45,47,53,57,71,75,93,96,99}
[AwardType, Selling] ----------> {0,2,6,7,8,15,22,33,34,37,44,52,54,76,77}
[AwardType, Producer] ----------> {0,2,5,7,8,10,17,22,33,34,42,44,47,53,54,57,71,75,77}
[Selling, Producer] ----------> {0,1,2,5,7,8,14,19,22,24,25,28,34,44,54,60,77,93,99}
[TypeID, MovieID, CountryCode] ----------> {0,1,4,7,8,12,17,21,22,28,29,35,38,50,99}
[TypeID, MovieID, MovCost] ----------> {0,1,7,19,21,22,28,29,35,44,52,54,60}
[TypeID, MovieID, Singer] ----------> {0,1,9,15,21,35,50,52,54,75}
[TypeID, MovieID, DirName] ----------> {0,1,7,12,15,17,19,21,22,28,42,60,77,99}
[TypeID, MovieID, Theatre] ----------> {0,7,9,12,17,22,38,42,44,52,54,75}
[TypeID, MovieID, Year] ----------> {0,7,8,9,12,17,22,42,44,52,75,77,93}
[TypeID, MovieID, AwardType] ----------> {0,7,8,12,17,21,22,35,38,44,52,60,75,99}
[TypeID, MovieID, Selling] ----------> {0,1,7,8,12,15,19,22,28,44,50,52,60,93}
[TypeID, MovieID, Producer] ----------> {0,1,7,8,17,19,22,28,29,42,44,54,60,75,93,99}
[TypeID, CountryCode, MovCost] ----------> {0,1,7,21,22,28,29,34,35,39,45,96}

[TypeID, CountryCode, Singer] ----------> {0,1,21,34,35,50,98}
[TypeID, CountryCode, DirName] ----------> {0,1,7,17,21,22,28,34,46,99}
[TypeID, CountryCode, Theatre] ----------> {0,7,17,22,25,34,38,39,46}
[TypeID, CountryCode, Year] ----------> {0,7,8,10,17,22,25,34,39}
[TypeID, CountryCode, AwardType] ----------> {0,7,8,17,21,22,34,35,38,45,46,99}
[TypeID, CountryCode, Selling] ----------> {0,1,7,8,22,28,34,50}
[TypeID, CountryCode, Producer] ----------> {0,1,7,8,10,17,22,25,28,29,34,45,99}
[TypeID, MovCost, Singer] ----------> {0,1,9,21,24,34,35,52,54}
[TypeID, MovCost, DirName] ----------> {0,1,7,19,21,22,24,28,38,42,60}
[TypeID, MovCost, Theatre] ----------> {0,7,9,22,24,34,38,42,44,52,54,73,76}
[TypeID, MovCost, Year] ----------> {0,7,9,22,24,38,42,44,52,76}
[TypeID, MovCost, AwardType] ----------> {0,7,21,22,34,35,38,44,45,52,60,76}
[TypeID, MovCost, Selling] ----------> {0,1,7,9,19,22,24,28,34,44,52,60,76}
[TypeID, MovCost, Producer] ----------> {0,1,7,9,19,22,24,28,29,34,42,44,45,54,60}
[TypeID, Singer, DirName] ----------> {0,1,2,15,21,24,38}
[TypeID, Singer, Theatre] ----------> {0,9,24,34,38,52,75}
[TypeID, Singer, Year] ----------> {0,9,24,38,52,75,89}
[TypeID, Singer, AwardType] ----------> {0,2,21,34,35,38,52,75}
[TypeID, Singer, Selling] ----------> {0,1,2,15,24,34,50,52,89}
[TypeID, Singer, Producer] ----------> {0,1,2,24,34,75}
[TypeID, DirName, Theatre] ----------> {0,7,17,22,24,34,38,42,46,91}
[TypeID, DirName, Year] ----------> {0,7,12,17,22,24,38,42,77,98}
[TypeID, DirName, AwardType] ----------> {0,2,7,12,17,21,22,34,38,46,60,99}
[TypeID, DirName, Selling] ----------> {0,1,2,7,15,19,22,24,28,34,60,91}
[TypeID, DirName, Producer] ----------> {0,1,2,7,17,19,22,24,28,34,60,99}
[TypeID, Theatre, Year] ----------> {0,7,9,12,17,22,24,25,28,39,42,44,52,75,76,89}
[TypeID, Theatre, AwardType] ----------> {0,7,12,17,22,34,38,44,46,52,75,76}
[TypeID, Theatre, Selling] ----------> {0,7,15,22,24,28,34,44,52,76,89,91}
[TypeID, Theatre, Producer] ----------> {0,7,17,22,24,25,28,34,44,75}
[TypeID, Year, AwardType] ----------> {0,7,8,12,17,22,34,38,44,52,75,76}
[TypeID, Year, Selling] ----------> {0,7,8,15,22,24,28,34,44,52,76,89,93}
[TypeID, Year, Producer] ----------> {0,7,8,10,17,22,24,25,28,34,44,75,93}
[TypeID, AwardType, Selling] ----------> {0,2,7,8,22,34,44,52,76}
[TypeID, AwardType, Producer] ----------> {0,2,7,8,17,22,34,44,75}
[TypeID, Selling, Producer] ----------> {0,1,2,7,8,19,22,24,25,28,34,44,54,60,93}
[MovieID, CountryCode, MovCost] ----------> {0,1,7,18,21,22,28,29,31,35,39,71,91,96}
[MovieID, CountryCode, Singer] ----------> {0,1,14,21,30,31,35,50,71}
[MovieID, CountryCode, DirName] ----------> {0,1,7,17,18,21,22,28,91,99}
[MovieID, CountryCode, Theatre] ----------> {0,7,17,18,22,38,39,91}
[MovieID, CountryCode, Year] ----------> {0,7,8,10,17,18,22,39,71}
[MovieID, CountryCode, AwardType] ----------> {0,7,8,17,18,21,22,31,35,38,71,99}
[MovieID, CountryCode, Selling] ----------> {0,1,6,7,8,22,28,50,91}
[MovieID, CountryCode, Producer] ----------> {0,1,7,8,10,17,22,28,29,71,99}
[MovieID, MovCost, Singer] ----------> {0,1,21,31,35,40,52,54,71}
[MovieID, MovCost, DirName] ----------> {0,1,7,18,19,21,22,28,60,91}
[MovieID, MovCost, Theatre] ----------> {0,7,18,22,44,49,52,54,91}
[MovieID, MovCost, Year] ----------> {0,7,18,22,44,49,52,53,71}
[MovieID, MovCost, AwardType] ----------> {0,6,7,18,21,22,31,35,44,52,60,61,71,97}
[MovieID, MovCost, Selling] ----------> {0,1,6,7,19,22,28,44,52,60,91}
[MovieID, MovCost, Producer] ----------> {0,1,7,19,22,28,29,44,53,54,60,71}
[MovieID, Singer, DirName] ----------> {0,1,2,3,15,21,91}
[MovieID, Singer, Theatre] ----------> {0,3,9,52,75,91}
[MovieID, Singer, Year] ----------> {0,9,52,71,75}
[MovieID, Singer, AwardType] ----------> {0,2,21,35,52,71,75}
[MovieID, Singer, Selling] ----------> {0,1,2,15,50,52,91}
[MovieID, Singer, Producer] ----------> {0,1,2,5,71,75,91}

[MovieID, DirName, Theatre] ----------> {0,3,6,7,17,18,22,42,91}
[MovieID, DirName, Year] ----------> {0,6,7,10,12,17,18,22,42,77,90}
[MovieID, DirName, AwardType] ----------> {0,2,6,7,12,17,18,21,22,60,99}
[MovieID, DirName, Selling] ----------> {0,1,2,6,7,15,19,22,28,60,91}
[MovieID, DirName, Producer] ----------> {0,1,2,5,6,7,10,17,19,22,28,57,60,99}
[MovieID, Theatre, Year] ----------> {0,5,7,9,12,17,18,22,39,42,44,49,52,57,75}
[MovieID, Theatre, AwardType] ----------> {0,5,6,7,12,17,18,22,31,38,44,52,75}
[MovieID, Theatre, Selling] ----------> {0,6,7,22,44,52,91}
[MovieID, Theatre, Producer] ----------> {0,5,7,17,22,44,57,75}
[MovieID, Year, AwardType] ----------> {0,6,7,8,12,17,18,22,31,44,52,71,75}
[MovieID, Year, Selling] ----------> {0,6,7,8,22,44,52,93}
[MovieID, Year, Producer] ----------> {0,5,6,7,8,10,17,22,44,47,53,57,71,75,93}
[MovieID, AwardType, Selling] ----------> {0,2,6,7,8,22,44,52}
[MovieID, AwardType, Producer] ----------> {0,2,5,7,8,10,17,22,44,53,71,75}
[MovieID, Selling, Producer] ----------> {0,1,2,7,8,19,22,28,44,60,93}
[CountryCode, MovCost, Singer] ----------> {0,1,21,31,34,35,52,54,71}
[CountryCode, MovCost, DirName] ----------> {0,1,7,18,21,22,28,37,42,60,91}
[CountryCode, MovCost, Theatre] ----------> {0,7,18,22,34,42,49,52,54,91}
[CountryCode, MovCost, Year] ----------> {0,7,18,22,42,49,52,71}
[CountryCode, MovCost, AwardType] ----------> {0,7,18,21,22,31,34,35,37,45,52,60,71}
[CountryCode, MovCost, Selling] ----------> {0,1,7,22,28,34,37,52,60,91}
[CountryCode, MovCost, Producer] ----------> {0,1,7,22,28,29,34,42,45,54,60,71}
[CountryCode, Singer, DirName] ----------> {0,1,21,37}
[CountryCode, Singer, Theatre] ----------> {0,34,52}
[CountryCode, Singer, Year] ----------> {0,52,71}
[CountryCode, Singer, AwardType] ----------> {0,21,34,35,37,52,71}
[CountryCode, Singer, Selling] ----------> {0,1,34,37,50,52}
[CountryCode, Singer, Producer] ----------> {0,1,34,71}
[CountryCode, DirName, Theatre] ----------> {0,7,17,18,22,34,42,46,54,91}
[CountryCode, DirName, Year] ----------> {0,7,17,18,22,42}
[CountryCode, DirName, AwardType] ----------> {0,7,17,18,21,22,34,37,46,60,99}
[CountryCode, DirName, Selling] ----------> {0,1,7,22,28,34,37,54,60,91}
[CountryCode, DirName, Producer] ----------> {0,1,7,17,22,28,34,54,57,60,99}
[CountryCode, Theatre, Year] ----------> {0,7,17,18,22,25,39,42,49,52,57}
[CountryCode, Theatre, AwardType] ----------> {0,7,17,18,22,34,38,46,52}
[CountryCode, Theatre, Selling] ----------> {0,7,22,34,52,91}
[CountryCode, Theatre, Producer] ----------> {0,7,17,22,25,34,57}
[CountryCode, Year, AwardType] ----------> {0,7,8,17,18,22,34,52,71}
[CountryCode, Year, Selling] ----------> {0,7,8,22,34,52}
[CountryCode, Year, Producer] ----------> {0,7,8,10,17,22,25,34,57,71}
[CountryCode, AwardType, Selling] ----------> {0,7,8,22,34,37,52}
[CountryCode, AwardType, Producer] ----------> {0,7,8,17,22,34,71}
[CountryCode, Selling, Producer] ----------> {0,1,7,8,22,28,34,54,60}
[MovCost, Singer, DirName] ----------> {0,1,21,24}
[MovCost, Singer, Theatre] ----------> {0,9,24,34,52}
[MovCost, Singer, Year] ----------> {0,9,24,52,71,94}
[MovCost, Singer, AwardType] ----------> {0,21,34,35,52,71,94}
[MovCost, Singer, Selling] ----------> {0,1,24,34,52}
[MovCost, Singer, Producer] ----------> {0,1,24,34,71}
[MovCost, DirName, Theatre] ----------> {0,7,18,22,24,38,42,91}
[MovCost, DirName, Year] ----------> {0,7,18,22,24,38,42,94,98}
[MovCost, DirName, AwardType] ----------> {0,7,18,21,22,37,38,60,64,94}
[MovCost, DirName, Selling] ----------> {0,1,7,19,22,24,28,37,60,91}
[MovCost, DirName, Producer] ----------> {0,1,7,19,22,24,28,32,57,60}
[MovCost, Theatre, Year] ----------> {0,7,9,18,22,24,42,44,49,52,57,64,76,94}
[MovCost, Theatre, AwardType] ----------> {0,7,18,22,34,38,44,52,64,76,94}

[MovCost, Theatre, Selling] ----------> {0,7,22,24,34,44,52,76,91}
[MovCost, Theatre, Producer] ----------> {0,7,22,24,34,44,57}
[MovCost, Year, AwardType] ----------> {0,7,18,22,38,44,52,64,71,76,94}
[MovCost, Year, Selling] ----------> {0,7,22,24,44,52,76,93}
[MovCost, Year, Producer] ----------> {0,7,22,24,44,53,57,71,93}
[MovCost, AwardType, Selling] ----------> {0,6,7,22,34,37,44,52,76}
[MovCost, AwardType, Producer] ----------> {0,7,22,34,44,57,71}
[MovCost, Selling, Producer] ----------> {0,1,7,19,22,24,28,34,44,60,93}
[Singer, DirName, Theatre] ----------> {0,3,18,24,38,39,91}
[Singer, DirName, Year] ----------> {0,18,24,38,39,77}
[Singer, DirName, AwardType] ----------> {0,2,18,21,37,38}
[Singer, DirName, Selling] ----------> {0,1,2,15,19,24,37,91}
[Singer, DirName, Producer] ----------> {0,1,2,19,24}
[Singer, Theatre, Year] ----------> {0,9,18,24,25,39,52,75,77}
[Singer, Theatre, AwardType] ----------> {0,18,34,38,52,75}
[Singer, Theatre, Selling] ----------> {0,24,34,52,91}
[Singer, Theatre, Producer] ----------> {0,24,25,34,75,77}
[Singer, Year, AwardType] ----------> {0,18,38,52,71,75,94}
[Singer, Year, Selling] ----------> {0,24,52,89}
[Singer, Year, Producer] ----------> {0,24,25,71,75}
[Singer, AwardType, Selling] ----------> {0,2,34,37,52}
[Singer, AwardType, Producer] ----------> {0,2,34,71,75}
[Singer, Selling, Producer] ----------> {0,1,2,19,24,25,34,77}
[DirName, Theatre, Year] ----------> {0,7,17,18,22,24,39,42,44,52,75,89}
[DirName, Theatre, AwardType] ----------> {0,6,7,17,18,22,31,34,38,44,46,52,75}
[DirName, Theatre, Selling] ----------> {0,6,7,22,24,34,44,52,89,91}
[DirName, Theatre, Producer] ----------> {0,7,17,22,24,34,44,75}
[DirName, Year, AwardType] ----------> {0,6,7,12,17,18,22,31,38,44,52,75,94,96}
[DirName, Year, Selling] ----------> {0,6,7,22,24,44,52,89,93}
[DirName, Year, Producer] ----------> {0,6,7,10,17,22,24,44,75,93,96}
[DirName, AwardType, Selling] ----------> {0,2,6,7,22,33,34,37,44,52}
[DirName, AwardType, Producer] ----------> {0,2,7,17,22,33,34,44,75}
[DirName, Selling, Producer] ----------> {0,1,2,7,14,19,22,24,28,34,44,54,60,93}
[Theatre, Year, AwardType] ----------> {0,7,8,12,17,18,21,22,35,44,45,52,61,64,71,75,76,94,99}
[Theatre, Year, Selling] ----------> {0,1,7,8,22,24,28,44,52,76,89}
[Theatre, Year, Producer] ----------> {0,1,5,7,8,17,22,24,25,28,44,45,47,57,71,75,99}
[Theatre, AwardType, Selling] ----------> {0,6,7,8,22,34,37,44,52,76}
[Theatre, AwardType, Producer] ----------> {0,5,7,8,17,22,34,44,71,75}
[Theatre, Selling, Producer] ----------> {0,1,7,8,22,24,28,34,44}
[Year, AwardType, Selling] ----------> {0,6,7,8,22,34,44,52,76}
[Year, AwardType, Producer] ----------> {0,7,8,17,22,34,44,71,75}
[Year, Selling, Producer] ----------> {0,1,7,8,22,24,28,34,44,93}
[AwardType, Selling, Producer] ----------> {0,2,7,8,22,34,44,54,77}
[TypeID, MovieID, CountryCode, MovCost] ----------> {0,1,7,21,22,28,29,35}
[TypeID, MovieID, CountryCode, Singer] ----------> {0,1,21,35,50}
[TypeID, MovieID, CountryCode, DirName] ----------> {0,1,7,17,21,22,28,99}
[TypeID, MovieID, CountryCode, Theatre] ----------> {0,7,17,22,38}
[TypeID, MovieID, CountryCode, Year] ----------> {0,7,8,17,22}
[TypeID, MovieID, CountryCode, AwardType] ----------> {0,7,8,17,21,22,35,38,99}
[TypeID, MovieID, CountryCode, Selling] ----------> {0,1,7,8,22,28,50}
[TypeID, MovieID, CountryCode, Producer] ----------> {0,1,7,8,17,22,28,29,99}
[TypeID, MovieID, MovCost, Singer] ----------> {0,1,21,35,52,54}
[TypeID, MovieID, MovCost, DirName] ----------> {0,1,7,19,21,22,28,60}
[TypeID, MovieID, MovCost, Theatre] ----------> {0,7,22,44,52,54}
[ActName, TypeID, MovieID, CountryCode, MovCost, DirName, Theatre, Year, AwardType, Selling, Producer] ----------> {7}

3. The given query responses from a community

(For simplicity & to save space, a few responses from communities are shown)

The given Query is:

SELECT MovieID,Year FROM movie WHERE MovieID="M110" OR Year="00"

The given query will hit in the following peers of Community:

{ 0,1,2,3,4,5,6,7,8,9,10,12,14,15,16,17,18,19,20,21,22,27,28,29,30,31,32,33,35,38,39,40,42,44,47,49,50,
52,53,54,55,57,60,61,62,65,70,71,75,77,81,82,85,90,91,93,96,97,99,11,24,25,34,45,56,63,64,66,69,76,79,
80,83,86,89,94,98 }

Response from Communities:

[MovieID] ----------> {0,1,2,3,4,5,6,7,8,9,10,12,14,15,16,17,18,19,20,21,22,27,28,29,30,31,32,33,35,38,
39,40,42,44,47,49,50,52,53,54,55,57,60,61,62,65,70,71,75,77,81,82,85,90,91,93,96,97,99}
[Year] ----------> {0,1,3,5,6,7,8,9,10,11,12,14,15,17,18,20,21,22,24,25,27,28,29,30,31,34,35,38,39,42,44,
45,47,49,52,53,55,56,57,60,61,62,63,64,65,66,69,71,75,76,77,79,80,81,82,83,85,86,89,90,93,94,96,98,99}
[TypeID, MovieID] ----------> {0,1,4,7,8,9,12,15,17,19,21,22,28,29,32,33,35,38,42,44,50,52,54,60,75,77,
81,93,99}
[TypeID, Year] ----------> {0,7,8,9,12,15,17,22,24,25,28,34,38,39,42,44,52,75,76,77,89,93,98}
[MovieID, CountryCode] ----------> {0,1,4,6,7,8,10,12,14,17,18,21,22,28,29,30,31,35,38,39,53,71,91,
96,99}
[MovieID, MovCost] ----------> {0,1,6,7,10,18,19,21,22,28,29,30,31,35,39,44,49,52,53,54,60,61,71,
91,96,97,99}
[MovieID, Singer] ----------> {0,1,2,3,5,9,10,14,15,21,31,35,40,50,52,54,71,75,91}
[MovieID, DirName] ----------> {0,1,2,3,5,6,7,10,12,15,16,17,18,19,21,22,28,40,42,50,57,77,90,91,99}
[MovieID, Theatre] ----------> {0,3,5,6,7,9,10,12,17,18,22,30,31,38,39,42,44,49,52,54,57,75,91}
[MovieID, Year] ----------> {0,3,5,6,7,8,9,10,12,14,17,18,20,22,30,31,39,42,44,47,49,52,53,55,57,71,75,
77,85,90,93}
[MovieID, AwardType] ----------> {0,2,5,6,7,8,10,12,17,18,21,22,31,35,38,40,44,52,53,55,60,61,71,75,
81,96,97,99}
[MovieID, Selling] ----------> {0,1,2,6,7,8,12,15,19,22,28,29,30,40,44,50,52,53,91,93}
[MovieID, Producer] ----------> {0,1,2,5,6,7,8,10,17,19,22,28,29,30,42,44,47,53,54,57,60,71,75,91,93,99}
[CountryCode, Year] ----------> {0,7,8,10,11,17,18,22,25,34,39,42,49,52,57,60,71,90}
[MovCost, Year] ----------> {0,7,9,14,18,22,24,38,42,44,49,52,53,57,64,71,76,80,83,90,93,94,98}
[Singer, Year] ----------> {0,9,18,24,25,30,38,39,52,71,75,77,89,94}
[DirName, Year] ----------> {0,6,7,11,12,14,17,18,22,24,31,38,39,42,44,52,60,65,75,77,89,90,93,94,
96,98}
[Theatre, Year] ----------> {0,1,5,7,8,9,12,17,18,21,22,24,25,28,30,35,39,42,44,45,47,49,52,57,61,64,71,
75,76,77,79,89,94,98,99}
[Year, AwardType] ----------> {0,6,7,8,10,12,17,18,21,22,31,34,35,38,44,45,52,60,61,64,71,75,76,81,94,
96,99}
[Year, Selling] ----------> {0,1,6,7,8,15,22,24,28,29,30,34,44,52,76,86,89,93}
[Year, Producer] ----------> {0,1,5,6,7,8,10,17,22,24,25,28,29,30,34,44,45,47,53,57,60,71,75,93,96,99}
[TypeID, MovieID, CountryCode] ----------> {0,1,4,7,8,12,17,21,22,28,29,35,38,99}
[TypeID, MovieID, MovCost] ----------> {0,1,7,19,21,22,28,29,35,44,52,54,60,99}
[TypeID, MovieID, Singer] ----------> {0,1,9,15,21,35,50,52,54,75}
[TypeID, MovieID, DirName] ----------> {0,1,7,12,15,17,19,21,22,28,42,50,77,99}
[TypeID, MovieID, Theatre] ----------> {0,7,9,12,17,22,38,42,44,52,54,75}
[TypeID, MovieID, Year] ----------> {0,7,8,9,12,17,22,42,44,52,75,77,93}

[TypeID, MovieID, AwardType] ----------> {0,7,8,12,17,21,22,35,38,44,52,60,75,81,99}
[TypeID, MovieID, Selling] ----------> {0,1,7,8,12,15,19,22,28,29,44,50,52,93}
[TypeID, MovieID, Producer] ----------> {0,1,7,8,17,19,22,28,29,42,44,54,60,75,93,99}
[TypeID, CountryCode, Year] ----------> {0,7,8,17,22,25,34,39}
[TypeID, MovCost, Year] ----------> {0,7,9,22,24,38,42,44,52,76}
[TypeID, Singer, Year] ----------> {0,9,24,38,52,75,89}
[TypeID, DirName, Year] ----------> {0,7,12,17,22,24,38,42,77,98}
[TypeID, Theatre, Year] ----------> {0,7,9,12,17,22,24,25,28,39,42,44,52,75,76,89}
[TypeID, Year, AwardType] ----------> {0,7,8,12,17,22,34,38,44,52,75,76}
[TypeID, Year, Selling] ----------> {0,7,8,15,22,24,28,34,44,52,76,89,93}
[TypeID, Year, Producer] ----------> {0,7,8,17,22,24,25,28,34,44,75,93}
[MovieID, CountryCode, MovCost] ----------> {0,1,7,10,18,21,22,28,29,30,31,35,39,71,91,96,99}
[MovieID, CountryCode, Singer] ----------> {0,1,10,14,21,31,35,71}
[MovieID, CountryCode, DirName] ----------> {0,1,7,10,17,18,21,22,28,91,99}
[MovieID, CountryCode, Theatre] ----------> {0,7,10,17,18,22,38,39,91}
[MovieID, CountryCode, Year] ----------> {0,7,8,10,17,18,22,39,71}
[MovieID, CountryCode, AwardType] ----------> {0,7,8,10,17,18,21,22,31,35,38,71,99}
[MovieID, CountryCode, Selling] ----------> {0,1,6,7,8,22,28,29,30,91}
[MovieID, CountryCode, Producer] ----------> {0,1,7,8,10,17,22,28,29,30,71,99}
[MovieID, MovCost, Singer] ----------> {0,1,10,21,31,35,52,54,71}
[MovieID, MovCost, DirName] ----------> {0,1,7,18,19,21,22,28,91,99}
[MovieID, MovCost, Theatre] ----------> {0,7,10,18,22,44,49,52,54,91}
[MovieID, MovCost, Year] ----------> {0,7,18,22,44,49,52,53,71}
[MovieID, MovCost, AwardType] ----------> {0,6,7,18,21,22,31,35,44,52,60,61,71,97,99}
[MovieID, MovCost, Selling] ----------> {0,1,6,7,19,22,28,29,30,44,52,91}
[MovieID, MovCost, Producer] ----------> {0,1,7,10,19,22,28,29,30,44,53,54,60,71,99}
[MovieID, Singer, DirName] ----------> {0,1,2,3,15,21,40,50,91}
[MovieID, Singer, Theatre] ----------> {0,3,9,10,52,75,91}
[MovieID, Singer, Year] ----------> {0,9,52,71,75}
[MovieID, Singer, AwardType] ----------> {0,2,21,35,52,71,75}
[MovieID, Singer, Selling] ----------> {0,1,2,15,40,50,52,91}
[MovieID, Singer, Producer] ----------> {0,1,2,5,10,71,75,91}
[MovieID, DirName, Theatre] ----------> {0,3,6,7,10,17,18,22,42,91}
[MovieID, DirName, Year] ----------> {0,6,7,12,17,18,22,42,77,90}
[MovieID, DirName, AwardType] ----------> {0,2,6,7,12,17,18,21,22,99}
[MovieID, DirName, Selling] ----------> {0,1,2,6,7,15,19,22,28,40,50,91}
[MovieID, DirName, Producer] ----------> {0,1,2,5,6,7,10,17,19,22,28,57,99}
[MovieID, Theatre, Year] ----------> {0,5,7,9,12,17,18,22,30,39,42,44,49,52,57,75}
[MovieID, Theatre, AwardType] ----------> {0,5,6,7,12,17,18,22,31,38,44,52,75}
[MovieID, Theatre, Selling] ----------> {0,6,7,22,30,44,52,91}
[MovieID, Theatre, Producer] ----------> {0,5,7,10,17,22,30,44,57,75}
[MovieID, Year, AwardType] ----------> {0,6,7,8,10,12,17,18,22,31,44,52,71,75}
[MovieID, Year, Selling] ----------> {0,6,7,8,22,30,44,52,93}
[MovieID, Year, Producer] ----------> {0,5,6,7,8,10,17,22,30,44,47,53,57,71,75,93}
[MovieID, AwardType, Selling] ----------> {0,2,6,7,8,22,40,44,52}
[MovieID, AwardType, Producer] ----------> {0,2,5,7,8,10,17,22,44,53,71,75}
[MovieID, Selling, Producer] ----------> {0,1,2,7,8,19,22,28,29,30,44,53,93}
[CountryCode, MovCost, Year] ----------> {0,7,18,22,42,49,52,71,90}
[CountryCode, Singer, Year] ----------> {0,52,71}
[CountryCode, DirName, Year] ----------> {0,7,17,18,22,42,90}
[CountryCode, Theatre, Year] ----------> {0,7,17,18,22,25,39,42,49,52,57}
[CountryCode, Year, AwardType] ----------> {0,7,8,10,17,18,22,34,52,60,71}
[CountryCode, Year, Selling] ----------> {0,7,8,22,34,52}
[CountryCode, Year, Producer] ----------> {0,7,8,10,17,22,25,34,57,60,71}

4. Query translation for community peers of a given query

Partial translated Query for a given Query:

Peer name	Translated query
1	FROM cinema WHERE (MovieName=Titanic OR MovieName=Bob the Builder OR MovieName=Lord of the Rin) OR (MovieYear=2000 OR MovieYear=2001 OR MovieYear=2002 OR MovieYear=2003 OR MovieYear=2004)
2	FROM film WHERE (MovieName=Terminator I)
3	FROM movie WHERE (MovieName=MacGyver OR MovieName=Alice in Wonder OR MovieName=Harry Potter)
4	FROM filmdesc WHERE (FilmID=F003) OR (FilmYear=01 OR FilmYear=02 OR FilmYear=04 OR FilmYear=95)
5	FROM cinema WHERE (CinemaName=Six Million $ma OR CinemaName=Queen Elizabeth) OR (CinemaYear=2000 OR CinemaYear=2001)
6	FROM moviedesc WHERE (MovID=Tita OR MovID=Home OR MovID=Lord) OR (MovieYear=2000 OR MovieYear=2001 OR MovieYear=2002 OR MovieYear=2003 OR MovieYear=2004)
7	FROM cinema WHERE (MovieName=Titanic OR MovieName=X Files OR MovieName=Dreamland)
8	FROM art WHERE (ArtName=Harry potter OR ArtName=McDonald OR ArtName=LandLord)
9	FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)
10	FROM movie WHERE (MovieName=Airforce II OR MovieName=Bob the Builder)
12	FROM movie WHERE (MovieName=Airforce II OR MovieName=Bob the Builder)
14	FROM movie WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)
15	FROM movie WHERE (MovieName=Airforce II OR MovieName=Bob the Builder)
16	FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder)
17	FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder)
18	FROM movie WHERE (MovieName=Airforce II OR MovieName=Bob the Builder)
19	FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder)
20	FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)
21	FROM movie WHERE (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)
22	FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)
27	FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)
28	FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)
29	FROM movie WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)
30	FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)
31	FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)
32	FROM movie WHERE (MovieName=Airforce II OR MovieName=Bob the Builder)
33	FROM movie WHERE (MovieName=Airforce II OR MovieName=Bob the Builder)
35	FROM film WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)
38	FROM film WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)

39 FROM movie WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
40 FROM cinema WHERE (MovieName=Airforce II)
42 FROM cinema WHERE (MovieName=Airforce II)
44 FROM art WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
47 FROM cinema WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
49 FROM film WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
50 FROM art WHERE (MovieName=Airforce II)
52 FROM art WHERE (MovieName=Airforce II)
53 FROM film WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
54 FROM movie WHERE (MovieName=Airforce II)
55 FROM film WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
57 FROM movie WHERE (MovieName=Airforce II)
60 FROM film WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
61 FROM movie WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
62 FROM film WHERE (MovieName=Airforce II)
65 FROM film WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
70 FROM movie WHERE (MovieName=Airforce II)
71 FROM film WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
75 FROM art WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
77 FROM art WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
81 FROM art WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
82 FROM film WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
85 FROM film WHERE (MovieName=Airforce II)
90 FROM movie WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
91 FROM film WHERE (MovieName=Airforce II)
93 FROM film WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
96 FROM cinema WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
97 FROM movie WHERE (MovieName=Airforce II)
99 FROM movie WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
24 FROM film WHERE (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)
34 FROM art WHERE (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)
45 FROM film WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
56 FROM art WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
63 FROM art WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
64 FROM cinema WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
66 FROM art WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
69 FROM film WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
76 FROM film WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
79 FROM cinema WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
80 FROM film WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
83 FROM art WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
86 FROM cinema WHERE (YearOfMovie=2000 OR YearOfMovie=2004)