University of Crete
School of Sciences and Engineering
Computer Science Department

TOP-K QUERY PROCESSING IN SCHEMA-BASED PEER-TO-PEER NETWORKS

by

IOANNIS C. CHRYSAKIS

Master's Thesis

Heraklion, March 2006

University of Crete
School of Sciences and Engineering
Computer Science Department

TOP-K QUERY PROCESSING IN SCHEMA-BASED PEER-TO-PEER NETWORKS

by

IOANNIS C. CHRYSAKIS

A thesis submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Author: _____
Ioannis Chrysakis, Computer Science Department

Supervisory
Committee: _____
Dimitris Plexousakis, Associate Professor, Supervisor

_____
Grigoris Antoniou, Professor, Member

_____
Dimitris Kotzinos , Adjunct Assistant Professor, Member

Approved by: _____
Dimitris Plexousakis, Associate Professor
Chairman of the Graduate Studies Committee

Heraklion, March 2006

TOP-K QUERY PROCESSING IN SCHEMA-BASED PEER-TO-PEER NETWORKS

IOANNIS C. CHRYSAKIS

MASTER THESIS

COMPUTER SCIENCE DEPARTMENT,
UNIVERSITY OF CRETE

ABSTRACT

The idea of Peer-to-peer (P2P) computing offers new opportunities for building highly distributed data systems. The advent of Semantic Web gave rise to a new category of peer-to-peer systems called Schema-Based. In Schema-Based P2P systems each peer is a whole database management system in itself. Each peer can use its own database schema, manages its own data and maintains its autonomy.

Considering a Schema-Based peer-to-peer network our main goal is the easy sharing of knowledge bases which implies efficient exchange of data across the p2p network without overly consuming bandwidth. For this reason, we first suggest a suitable peer-to-peer architecture and a well defined query routing context. Our main contribution is the proposal of a query routing strategy and a query processing strategy. The proposed query routing strategy directs the query to a set of relevant peers in such way as to avoid network traffic and bandwidth consumption.

Our processing technique is based on the idea of top-k queries that has arisen in database research. Simply top-k queries return only the k best results according to a given criterion. Recently top-k retrieval algorithms for distributed networks have been presented following different approaches. After presenting these approaches and determining their advantages and drawbacks, we conclude that the Hybrid Threshold (HT) algorithm could be the best solution for top-k processing in peer-to-peer networks. We extend HT and adapt it under our well-defined peer-to-peer environment, and in consequence we suggest two improved versions: HT-p2p and HT-p2p+. The first assumes that results are returned by executing an instance of the algorithm to a specified Super-Peer, named collector Super-Peer. The latter assumes that results come from the combination of all top-k object sets that are

returned from each running instance of the algorithm to each specified contributor Super-Peer. In addition, since HT-p2p belongs to score-based top-k algorithms we study the problem of scoring objects and suggest accordingly three use cases of the algorithm.

For the evaluation of HT-p2p and HT-p2p+ we implement a prototype system built upon the JXTA platform. The results of the experiments upon HT-p2p system showed that our proposed algorithm is a scalable, and efficient top-k processing algorithm that could be used by any Super-Peer based peer-to-peer network.

Supervisor: Dimitris Plexousakis
Associate Professor

ΕΠΕΞΕΡΓΑΣΙΑ Κ-ΚΟΡΥΦΑΙΩΝ ΕΡΩΤΗΣΕΩΝ ΣΕ ΟΜΟΤΙΜΑ ΔΙΚΤΥΑ

ΙΩΑΝΝΗΣ Κ. ΧΡΥΣΑΚΗΣ

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ,
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

ΠΕΡΙΛΗΨΗ

Τα ομότιμα δίκτυα (peer-to-peer (P2P) networks) παρέχουν πολλές δυνατότητες για την ανάπτυξη πλήρως κατανεμημένων συστημάτων διαχείρισης δεδομένων. Καθώς η ιδέα του σημασιολογικού ιστού άρχισε να εδραιώνεται, έκαναν την εμφάνιση τους τα ομότιμα συστήματα στα οποία κάθε κόμβος διαχειρίζεται μια ξεχωριστή βάση δεδομένων και για την οποία διατηρεί ένα συγκεκριμένο σχήμα. (Schema-Based peer-to-peer networks).

Θεωρώντας ένα Schema-Based peer-to-peer network βασικός στόχος μας είναι ο εύκολος διαμοιρασμός της πληροφορίας με το ελάχιστο εύρος των δεδομένων που πρέπει να μετακινηθούν κατά μήκος του ομότιμου δικτύου. Για τον λόγο αυτό προτείνουμε μια κατάλληλη αρχιτεκτονική για το συνιστώμενο ομότιμο δίκτυο και ένα καλά ορισμένο πλαίσιο δρομολόγησης των ερωτήσεων. Η κεντρική συνεισφορά της εργασίας έγκειται στην πρόταση μιας ολοκληρωμένης στρατηγικής δρομολόγησης και επεξεργασίας της κάθε ερώτησης. Η προτεινόμενη στρατηγική δρομολόγησης αναλαμβάνει την κατεύθυνση της ερώτησης στους κατάλληλους κόμβους χωρίς να δημιουργεί αρκετή κυκλοφορία στο ομότιμο δίκτυο γεμίζοντας το με άσκοπα μηνύματα.

Η προτεινόμενη στρατηγική επεξεργασίας βασίζεται στην ιδέα των κ-κορυφαίων ερωτήσεων η οποία πρωτοεμφανίστηκε στον τομέα των βάσεων δεδομένων. Οι κ-κορυφαίες ερωτήσεις επιστρέφουν τα κ καλύτερα αποτελέσματα δεδομένου κάποιου ορισμένου κριτηρίου. Πρόσφατα αυτή η ιδέα άρχισε να εφαρμόζεται σε κατανεμημένα δίκτυα. Αφού παρουσιάσουμε και αναλύσουμε τις υπάρχουσες προσεγγίσεις συμπεραίνουμε ότι ο υβριδικός αλγόριθμος (HT) ταιριάζει καλύτερα στο δικό μας σενάριο, γι αυτό τον επεκτείνουμε και τον προσαρμόζουμε στις απαιτήσεις του συγκεκριμένου κατανεμημένου περιβάλλοντος. Τελικά παρουσιάζουμε δύο εκδόσεις του βελτιωμένου μας αλγορίθμου (HT-p2p, HT-p2p+) ανάλογα

με την περίπτωση χρήσης του. Επιπλέον, δεδομένου ότι ο αλγόριθμος ανήκει στην οικογένεια των βασιζόμενων σε σκορ αλγορίθμων, μελετάμε το πρόβλημα της βαθμολόγησης των αντικειμένων και προτείνουμε τρία σενάρια χρήσης για κάθε περίπτωση.

Για την αποτίμηση του HT-p2p αλγορίθμου αναπτύξαμε ένα σύστημα χρησιμοποιώντας την τεχνολογία που παρέχει η πλατφόρμα JXTA. Τα αποτελέσματα των πειραμάτων έδειξαν ότι η προτεινόμενος αλγόριθμος έχει καλή κλιμακωσιμότητα και είναι αποδοτικός σε κάθε ομότιμο δίκτυο που ακολουθεί την προτεινόμενη αρχιτεκτονική μας.

Επόπτης: Δημήτρης Πλεξουσάκης
Αναπληρωτής Καθηγητής

Στους γονείς μου Κώστα και Μαρία και στην αδερφή μου Εύα
για την στήριξη τους σε κάθε μου προσπάθεια...

Ευχαριστίες

Σε αυτό το σημείο θα ήθελα καταρχήν να ευχαριστήσω τον επόπτη μου κ. Δημήτρη Πλεξουσάκη που μου έδωσε την ευκαιρία να ασχοληθώ με ένα επίκαιρο και σίγουρα ενδιαφέρον θέμα. Οι αμέτρητες συζητήσεις που είχαμε μαζί, οι ουσιαστικές υποδείξεις και οι πολύτιμες συμβουλές του συνετέλεσαν στην ουσιαστική καθοδήγηση και πρόοδο κατά την εκπόνηση της παρούσας εργασίας. Ακόμη, σε δύσκολες στιγμές η κατανόηση του και η διάθεση του για συζήτηση έλυνε κάθε πρόβλημα που φάνταζε αδιέξοδο.

Επίσης ευχαριστώ θερμά τον καθηγητή μου κ. Δημήτρη Κοτζίνο με τον οποίο είχα την τιμή να συνεργαστώ κατά τη διάρκεια του μεταπτυχιακού προγράμματος και να αποκομίσω σημαντικές γνώσεις καθώς και να γνωρίσω ένα ιδιαίτερα οργανωτικό τρόπο δουλειάς. Οι πολύτιμες επισημάνσεις του αναμφισβήτητα βελτίωσαν την παρούσα εργασία.

Παράλληλα θα ήθελα να ευχαριστήσω ιδιαιτέρως τον κ. Γρηγόρη Αντωνίου για την προθυμία του να συμμετάσχει στην εξεταστική επιτροπή μου και για τις εύστοχες παρατηρήσεις του.

Ένα μεγάλο ευχαριστώ αξίζει στον Γιάννη Καπανταϊδάκη, με τον οποίο μοιραστήκαμε πολλές ώρες συζητήσεων σε ερευνητικά και μη ζητήματα. Η συμβολή του στην κατανόηση πολύπλοκων ζητημάτων ήταν καθοριστική. Επίσης η προθυμία του για βοήθεια ήταν και παραμένει αξιόλογη.

Επιπλέον, ευχαριστώ τον Νίκο Δημόκα, τον Νίκο Παπαδόπουλο και τον Νίκο Ξανθόπουλο που έλυσαν πλήθος απορίων μου.

Ένα μεγάλο ευχαριστώ επίσης ανήκει σε όλους τους φίλους/ες μου με τους οποίους μοιραστήκαμε κοινές προσπάθειες και εμπειρίες. Κυρίως όμως θέλω να ευχαριστήσω την Ρίτα Πετράκη για την κατανόηση, συμπαράσταση και την εμψύχωση καθ’ όλη τη διάρκεια της μεταπτυχιακής μου εργασίας.

Επίσης, ευχαριστώ το Τμήμα Επιστήμης Υπολογιστών του Πανεπιστημίου Κρήτης και την Ομάδα Πληροφορικών Συστημάτων του Ινστιτούτου Πληροφορικής για όλα όσα μου προσέφεραν όλα αυτά τα χρόνια, για τις γνώσεις και για τις ανεκτίμητες εμπειρίες που αποκόμισα μέσα σε ένα ιδιαίτερα φιλικό περιβάλλον εργασίας.

Κλείνοντας θα ήθελα πάνω από όλα να ευχαριστήσω τους γονείς μου Κώστα και Μαρία καθώς και την αδερφή μου Εύα για την αμέριστη συμπαράσταση τους σε κάθε φάση της ζωής μου.

Γιάννης Χρυσάκης

Table of Contents

INTRODUCTION **1**

1.1 MOTIVATION 1
1.2. OBJECTIVES 3
1.3 THESIS CONTENTS 5

BACKGROUND AND RELATED WORK **6**

2.1 PEER-TO-PEER NETWORKS 6
2.1.1 Peer-to-peer in general 6
2.1.2 Peer-to-peer classification 7
2.1.3 Schema-Based P2P networks 8
2.2 SEMANTIC WEB STANDARDS AND TECHNOLOGIES 9
2.2.1 Semantic Web in general 9
2.2.2 XML 10
2.2.3 RDF/S 11
2.2.4 OWL 12
2.3 QUERY ROUTING IN P2P NETWORKS 12
2.3.1 The Problem of Query Routing 12
2.3.2 Query Routing Based on Routing Indices and Super-Peer network 13
2.3.3 Query Routing Based on SOCs and Information Peer Models on Super-Peer networks 17
2.3.4 Routing Indices and/or SOC Approach 20
2.3.5 Query Routing Based on Social Metaphors and Shortcut Indexes 21
2.3.6 Query Routing Based on Query Patterns 24
2.4 QUERY PROCESSING IN P2P NETWORKS 27
2.4.1 The Problem of Query Processing 27
2.4.2 Query Processing based on Query Planning and Optimization 28
2.4.3 Query Processing for top-k queries 30
2.4.3.1 Top-k queries in general and P2P networks 30
2.4.3.2 Assumptions on existing approaches of top-k query processing in P2P networks 31
2.4.3.3 The Probabilistic – Histograms Approach 31
2.4.3.4 The Nejdl et. al Approach 32
2.4.3.5 The Marian et. al Approach 34
2.4.3.6 Three Phase Threshold Approach 36
2.5 CHAPTER SUMMARY 39

METHODOLOGY **41**

3.1 BASIC CONTEXT AND DIRECTIONS 41
3.1.1 Formulation of the problem 41
3.1.2 Design Decisions for Peer-to-peer networks 42
3.2 QUERY ROUTING STRATEGY 44
3.2.1 Basic features and query routing context 44
3.2.2 Query Routing Algorithm 46
3.2.3 Advantages of Query Routing Strategy 48
3.3 QUERY PROCESSING STRATEGY 49
3.3.1 General Features, Issues and basic steps 49
3.3.2 Selection of Top-k basic algorithm 50
3.3.3 The Hybrid Threshold Algorithm (HT) 51
3.3.3.1 The original applied context 51
3.3.3.2 The original HT 52
3.3.3.3 Evaluation of HT 54
3.3.4 The HT-p2p: A Hybrid Threshold algorithm for a Super-Peer –Based P2P 56
3.3.4.1 The HT-p2p context and basic features 56
3.3.4.2 The HT-p2p Algorithm 57
3.3.4.3 The HT-p2p+ Algorithm 61
3.3.4.4 Data Scoring and Use Cases of HT-p2p 62
3.3.4.5 Cost Analysis of HT-p2p / HT-p2p+ 63

3.3.4.6 An example of HT-p2p *65*
3.3.5 Advantages of Query Processing Strategy *67*
3.4 CHAPTER SUMMARY 68

IMPLEMENTATION 69

4.1 JXTA TECHNOLOGY 69
4.1.1 Definition and Objectives 69
4.1.2 JXTA Architecture and Protocols 71
4.1.3 JXTA Basic Concepts 73
4.1.3.1 Identifiers (IDs) *73*
4.1.3.2 Peers *74*
4.1.3.3 Peer Groups *74*
4.1.3.4 Advertisements *75*
4.1.3.5 Messages *76*
4.1.3.6 Pipes *77*
4.2 THE HT-P2P SYSTEM 78
4.2.1 System Description, Design Decisions and Basic Features 78
4.2.2 System Design and Architecture 81
4.2.3 Communication Module 82
4.2.3.1 Implementation Decision *82*
4.2.3.2 Basic Functionality *83*
4.2.4 Super-Peer Module 84
4.2.5 Peer Module 87
4.3 CHAPTER SUMMARY 88

EVALUATION 89

5.1 EXPERIMENTAL SETUP 89
5.2 EXPERIMENTS 90
5.2.1 Experiment 1 90
5.2.2 Experiment 2 92
5.2.3 Experiment 3 95
5.3 CHAPTER SUMMARY 96

CONCLUSIONS 97

6.1 SUMMARY 97
6.2 EXTENSIBILITY SUGGESTIONS 100
6.2.1 Suggestions for Query Routing Strategy 100
6.2.2 Suggestions for Query Processing Strategy 101
6.3 CHAPTER SUMMARY 102

REFERENCES 103

List of Figures

Figure 1: The Semantic Web Tower 9
Figure 2: Super-peer/peer routing index 14
Figure 3: Super-peer/super-peer routing index 15
Figure 4: A routing example based on Routing Indices 16
Figure 5: SP/SP index of SP_2 at different granularities 16
Figure 6: Super-Peer Network with Clustering Policy Information Provider Model 19
Figure 7: Matching and distribution of models in the Super-Peer Network 20
Figure 8: Shortcut overlay and roles of peers 23
Figure 9: RDF/S schema namespace, peer active-schema and query pattern graph 25
Figure 10: SQPeer Query Routing Algorithm 27
Figure 11: Query Planning Generation at Super-Peers 28
Figure 12: The pUpper Algorithm 35
Figure 13: An example of two taxonomies 45
Figure 14: An example of a P2P network built upon our proposed architecture and with regard to our suggested routing context 47
Figure 15: Basic abstract steps to processing of a query 50
Figure 16: Performance comparisons over a synthetic data set 55
Figure 17: The JXTA three-layer architecture 72
Figure 18: JXTA specification protocols hierarchy 73
Figure 19: An example of a pipe advertisement 76
Figure 20: JXTA Configurator Basic Settings 80
Figure 21: JXTA Configurator Advanced Settings 80
Figure 22: The building blocks of HT-p2p's architecture 81
Figure 23: Bidirectional connection through pipe for message transfer 83
Figure 24: Multi-Threaded Architecture of Super-Peer 84
Figure 25: Execution time as k increases in HT-p2p 91
Figure 26: Execution time as contributor peers are increased in HT-p2p (use of random scoring function) 93
Figure 27: Execution time as contributor peers are increased in HT-p2p (use of same scoring function) 94

List of Tables

Table 1: Rules within a clustering policy 18
Table 2: Required messages for a completed executing scenario of HT-p2p+..... 64
Table 3: (Object, Score) pairs at each peer of SP1 65
Table 4: Results Table of Experiment 1..... 90
Table 5: Results Table of Experiment 2a..... 93
Table 6: Results Table of Experiment 2b..... 94

Chapter 1

Introduction

1.1 Motivation

The idea of peer-to-peer (P2P) computing offers new opportunities for building highly distributed data systems. Specifically, the P2P computing provides a very efficient way of storing and accessing distributed resources. Peer-to-peer systems are distributed systems without any centralized control in which each node shares and exchange data across the network (peer-to-peer network). A review of the features of recent peer-to-peer systems yields a long list: redundant storage, permanence, selection of nearby servers, anonymity, search, authentication, and hierarchical naming. They also offer the potential for low cost sharing of information, autonomy and privacy since they take advantage of decentralization by distributing the storage, information and computation cost among the peers. In addition to the ability to pool together and harness large amounts of resources, the strengths of existing P2P systems include self-organization, load-balancing, adaptation, and fault tolerance.

Using peer-to-peer systems for the exchange of files, especially of music files, is a quite common application. Examples of such systems are Napster [1], Gnutella [2], Freenet [3], Morpheus [4] and Kazaa [5]. Despite the recent emergence of P2P systems, most of these systems have severe limitations in contrast to traditional data management systems: file-level sharing, read-only access, simple keyword-based search and poor scaling. In most cases, searching in a P2P system relies on simple selection conditions on a predefined set of document attributes or IR-style string matching. Simple techniques (e.g., network flooding) are used to lookup and retrieve relevant data. Moreover, both communication and processing resources are wasted due to the fact that no optimizations are usually considered. These limitations may be acceptable for file-sharing applications, but in order to support highly dynamic, ever-changing, autonomous social organizations (e.g., scientific or educational communities) we need richer facilities in exchanging, querying and integrating

structured and semi-structured data hosted by peers. Moreover, considering data management issues in P2P systems is a quite challenging task due to the scale of the network and the autonomy and unreliable nature of peers.

Some work has been done to support some critical data management issues in P2P systems. Thus, recently peer-to-peer networks have also been used successfully to interconnect between distributed heterogeneous scientific data stores enabling the exchange of scientific documents and the search in complex heterogeneous meta-data structures. Examples for this new class of peer-to-peer networks, so called *Schema Based peer-to-peer networks,* are [6, 7, 8, 9]. Such networks combine approaches from peer-to-peer research as well as from the database and semantic web research areas. The combination of *Semantic Web* and peer-to-peer technologies, i.e., the use of semantic descriptions of data sources stored by peers and of semantic descriptions of the peers themselves, is claimed to help in formulating queries in such a way that they can be understood by other peers, in merging the answers received from different peers, and in directing queries across the network. Thus, Schema-Based P2P networks allow the aggregation and integration of data from autonomous, distributed data sources. They build upon peers that use explicit schemas to describe their content. Naturally such metadata is pretty heterogeneous as documents stem from a wide variety of domains and communities.

However current Schema-Based peer-to-peer networks still have some shortcomings. In their beginning, Schema-Based P2P networks broadcast all queries to all peers so, their scalability is limited. Intelligent routing and network organization strategies are essential in such networks so queries are only routed to a semantically chosen subset of peers able to answer parts or whole queries. First approaches to enhance routing efficiency in a clustered network have already been proposed by [10] and [11]. Semantic Overlay Networks as they presented in [10] is a fundamental concept where query routing can be build. However it was not showed how these networks can be used practically in a Schema-Based peer-to-peer network. Also the peer clustering and firework query model [11] as it was presented is based on Information Retrieval models, and its applicability to Schema-Based P2P networks is difficult and inauspicious.

Recently, the problem of efficient query routing in a (Schema-Based) Peer to Peer Network has been studied by some authors. All the dominant approaches are

presented in the background section, and it is analyzed their applied peer-to-peer environment. All of them have advantages and disadvantages. We examine in detail all of these approaches in the next section in order to draw conclusions that lead us to our suggested approach under our defined peer-to-peer context.

Query processing in peer-to-peer network is a multifaceted topic in which many authors have studied. A variety of techniques have been suggested following varying hypotheses and addressing different aspect of the problem. But for Schema-Based peer to peer networks there is a minimal work [12, 13] in query processing of such systems that is based on query planning and optimization. These techniques use a restricted query model and seem to introduce a large processing cost to each peer, in order to get back the results.

Hence, efficient ways of query processing must be also supported in order to gain fast retrieval of data and without large bandwidth consumption the results of each query across the network. The idea of *top-k queries* was first introduced in [14] and applied to relational databases. Simply put, top-k queries return only the k best results according to a given criterion. Generally, top-k queries on multidimensional datasets compute the k most relevant or interesting results to a partial-match query, based on similarity scores of attribute values with regard to elementary query conditions and a score aggregation function such as weighted summation. Bearing in mind that the P2P systems are designed to build global-scale information systems, it is quite easy for a user to obtain a huge amount of results in response to a given query. Thus, it is obviously important to support top-k queries in order to contribute to a good overall performance of the P2P system, since it provides quality, filtering on results and an effective solution when we don't have an exact match.

1.2. Objectives

In this work we propose a complete framework for efficient query routing and processing in a P2P network. This framework can be supported by any Schema-Based peer to peer network. To enable the harmonic combination of our query routing and processing strategy we propose an architecture that takes advantage of the characteristics of our proposed strategies under a well defined peer-to-peer network.

This architecture is based on super-peers and suggests an unstructured (hybrid) peer-to-peer network.

Each peer in this peer-to-peer network manages an autonomous local knowledge base on some given subjects (e.g. Computer software, Internet) independently of the other peers. The knowledge base is represented as a simple hierarchy of terms, each term representing a topic of interest. Our routing technique exploits the assumption that each peer has its own taxonomy of terms that describes its schema and there are two-way links between terms of different knowledge bases. These taxonomies are published to corresponding super-peers which have the responsibility of query routing. Thus, each peer which is connected to the network shares its knowledge with other peers by making queries on it. Finally, the query is routed to suitable peers from corresponding super-peers in such way to avoid network traffic and bandwidth consumption.

Our processing technique is based on the idea of top-k queries that has arisen in database research. For distributed networks this idea has been applied to minimal efforts. In this work we compare all these efforts that promise to apply to peer-to-peer networks and choose the more efficient top-k retrieval algorithm. This algorithm is *Hybrid Threshold (HT)* which is introduced by [15]. We adapt this algorithm under our peer-to-peer environment and improve it by pruning two phases under certain conditions. Also we extend HT in order to use it by many contributor peers and their responsible Super-Peers and finally adapt in our proposed peer-to-peer environment. Our suggested improved Hybrid Threshold algorithm has been named *HT-p2p*. Moreover considering a more distributed scenario for large number of contributor peers at different responsible Super-Peers, we present a modified version of HT-p2p called *HT-p2p+*. In addition, since HT-p2p belongs to score-based top-k algorithms we study the problem of scoring objects and suggest accordingly three use cases of the algorithm.

To access the efficiency of our proposal we implemented a system that uses these algorithms. The system was designed on top of the JXTA platform [16]. JXTA is an open network computing platform designed for peer-to-peer computing. It provides a common set of open protocols and an open source reference implementation for developing general purpose, interoperable and large scale P2P applications.

The system consists of peers and Super-Peers. Each category of peers has its own methods, functionality and contribution to the computation of top-k results. Using our implementation we conducted experiments in order to test this algorithm under realistic conditions of a peer-to-peer network. The results showed that our proposed processing strategy is efficient enough for super-peer based network. Also the results show us the way to improve HT-p2p by suggesting and implementing some extensions. Finally this work suggests how a system that exchanges data across a distributed environment may fulfil the important demands of its users: fast query answering, easy data sharing, stability, privacy, self organizing, autonomy and load-balancing.

1.3 Thesis Contents

The thesis is structured as follows:

Chapter 2 presents the main concepts that are used in our work and discusses the related work to the specific domain of query routing and processing in peer-to-peer environments.

Chapter 3 describes our suggested methodology for efficient query routing and processing in Schema-Based peer-to-peer networks. In particular this chapter describes our suggested query routing technique and our proposed top-k query processing strategy.

Chapter 4 describes the prototype system HT-p2p which implements our suggested top-k query processing strategy and is built upon the JXTA platform. In this way we show in practise how HT-p2p algorithm can be used by any Super-Peer based peer-to-peer network.

Chapter 5 presents a set of experiments that we performed in order to evaluate the basic characteristics of our two algorithms (HT-p2p, HT-p2p+). We also discuss the results and arrive to important conclusions regarding the performance of the algorithms and their extensibility.

Chapter 6 summarizes our work and its contributions and presents some extensibility suggestions for our query routing technique and our top-k query processing strategy.

Chapter 2

Background and Related Work

In this chapter, we present the main concepts that are used in our work. We start from the basic ones like peer-to-peer networks and Semantic Web and finally we present the related work that has been done in the specific domain of query routing and processing in p2p environments which are the focus of this thesis.

2.1 Peer-to-peer networks

2.1.1 Peer-to-peer in general

During the last years, peer-to-peer (P2P) systems have seen resurgence. The idea of autonomous coequal nodes fulfilling a certain task without any central coordinator dates back to the very first designs of the ARPA net. After being eclipsed by client/server architectures, peer-to-peer systems regained attention as highly scalable file sharing platforms during the last decade. In a peer-to-peer network the nodes which are called peers are designed equal and are considered to be autonomous. Each peer can act as both client and server.

The main advantage of peer-to-peer networks is that they distribute the responsibility of providing services among all peers on the network. This fact eliminates service outages due to a single point of failure and provides a more scalable solution for offering services. Moreover, P2P networks exploit available bandwidth across the entire network by using a variety of communication channels and by filling bandwidth to the "edge" of the Internet.

Unlike traditional client/server communications, in which specific routes to popular destinations can become overtaxed, peer-to-peer enables communication via a variety of network routes, thereby reducing network congestion. Peer-to-peer has the capability of serving resources with high availability at a much lower cost while maximizing the use of resources from every peer connected to the peer-to-peer network. Whereas client/server solutions rely on the addition of costly bandwidth,

equipment, and co-location facilities to maintain a robust solution, peer-to-peer can offer a similar level of robustness by spreading network and resource demands across the network.

Hence, P2P promotes the sharing of resources and services through direct exchange between peers. Resources can be processing cycles (SETI@home), collaborative work (ICQ, Waste), storage space (Freenet), network bandwidth (ad hoc networking, internet) or data. Large scale information systems are built upon a peer-to-peer network where each peer exchanges its data. The last is the most famous, utilizable and useful case which we have dealt within this work.

2.1.2 Peer-to-peer classification

There are two main categories in which we can classify between peer-to-peer systems: structured and unstructured. The structured peer-to-peer systems distribute data across the network according to a hash function, in order to form a distributed hash table (DHT). Thus each peer holds a data structure that maintains information about what data is available via each of its neighbours. Examples for structured P2P systems are Chord [17] and CAN [18]. Chord is a ring-based system, whereas CAN maps the key space on a torus. At this category of peer-to-peer we gain fast retrieval of data (O(logn)). However, their disadvantage is that they support only key lookup queries and range queries which limit the query capabilities of the whole P2P system.

On the other hand in unstructured peer-to-peer systems peers are free to manage their own data. The ancestors of P2P Napster [1] and Gnutella [2] are representative of this category of systems. Unstructured peer-to-peer systems can support rich query languages. Data is found either by maintaining a centralized index, or by flooding with messages the whole network. Hence, to gain fast and successful retrieval of data there is need for efficient query routing and processing techniques.

Another classification of P2P systems is based on network topology. Thus we talk about pure peer-to-peer networks and super-peer networks. In pure P2P networks peers don't follow a specific topology as they join in the network, so we have full distributed and independent peers. In fact, at these networks all peers are equivalent, namely they have the same role and responsibilities. Thus, there is no centralized server. But for all these above reasons the flooding of messages and bottlenecks

across the pure peer-to-peer network is a frequent phenomenon. Gnutella [2] also belongs to this category.

Super-Peer-Based P2P networks combine the efficiency of a centralized search (super peers route the query to appropriate peers) with the autonomy, load balancing and robustness to attacks provided by distributed search. A super-peer is a node of the network that acts as a server to a subset of clients. This network topology takes advantage the heterogeneity of peers and it is scalable as new peers join. KazaA [5] is a well-known super-peer system. A new class of P2P systems called Schema-Based appeared recently and combine approaches from peer-to-peer research, as well as, from the database and semantic web research areas. We denote the main characteristics of this class in the next subsection since in our work we focus on the semantic exchange of information across the distributed network.

2.1.3 Schema-Based P2P networks

In Schema-Based peer-to-peer systems each peer is a whole database management system in itself. The system manages its own data and maintains its autonomy. Moreover, each peer can use its own database schema. As neighbouring peers may have different schemas, these have to be mapped when peers exchange data or query requests. Therefore, links do not only represent a means of data exchange but they are also used for data integration. Thus, this approach promises to keep costs low for the important problem of data integration.

The semantic web standards such as XML, RDF, and OWL are helpful in this direction of easy data integration. Also, their adaptation to Schema-Based systems has the advantage of knowledge reuse, easy schema creation, manipulation and navigation. In addition to these standards there is a support of rich and functional query languages upon these schemas. The combination of Semantic Web and Peer-to-Peer technologies results in building large scale peer-to-peer systems that formulate queries in such a way that can be understood by other peers, merging the answers received from different peers, and directing queries across the network Finally we have to note that, Schema-Based P2P networks can be built on any network topology (pure, super-peer) and they can be structured (using DHT) or unstructured. Our design decisions for our peer-to-peer network are analyzed in the next chapter.

2.2 Semantic Web Standards and Technologies

2.2.1 Semantic Web in general

The Web has been created as a source of information for humans. For many, this medium has become indispensable. As a vision for the future, the Web could and should be extended with information that can be understood by machines. This would be the foundation for a new class of applications, and would also result in the improved interconnectivity of available information. This new kind of Web called Semantic Web [19] aims for machine-interpretable Web resources, whose information can be shared and processed both by automated tools, such as search engines, and human users. Semantic Web is a collaborative effort led by W3C with participation from a large number of researchers and industrial partners. The development of the Semantic Web proceeds in steps, each step building a layer on top of another. The layered design is shown in Figure 1 below.

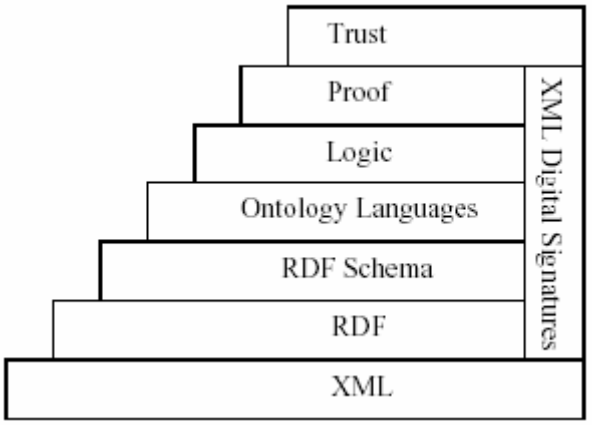

Figure 1: The Semantic Web Tower

At the bottom layer we find XML, a language that lets one write structured web documents with a user-defined vocabulary. XML is particularly suitable for sending documents across the Web, thus supporting syntactic interoperability. RDF is a basic data model, like the entity-relationship model, for writing simple statements about Web objects (resources). The RDF data model does not rely on XML, but RDF has an XML-based syntax. Therefore, it is located on top of the XML layer. RDF Schema provides modeling primitives, for organizing Web objects into hierarchies. RDF Schema is based on RDF. RDF Schema can be viewed as a primitive language for writing ontologies. But there is a need for more powerful ontology languages that

expand RDF Schema and allow the representations of more complex relationships between Web objects. Ontology languages, such as OWL, are built on the top of RDF and RDF Schema.

The logic layer is used to enhance the ontology language further, and to allow writing application-specific declarative knowledge. The proof layer involves the actual deductive process, as well as the representation of proofs in Web languages and proof validation. Finally, trust will emerge through the use of digital signatures, and other kind of knowledge, based on recommendations by agents we trust, or rating and certification agencies and consumer bodies.

XML, RDF/S and OWL can be easily used for Schema-Based peer-to-peer systems. One example of a Schema-Based P2P that is based on RDF/S is Edutella [20]. In the Edutella network every peer needs to make its metadata available as a set of RDF statements that rely on a certain schema. These three basic standards are analyzed further in the next subsections. For a comprehensive completed description to the Semantic Web and its basic standards refer to [21].

2.2.2 XML

XML [22] stands for eXtensible Markup Language and it is the universal format for structured documents and data on the Web. The success of XML is primarily based on its flexibility since everybody can write a document type definition (DTD) or XML Schema to define the structure of XML documents that represent information in the form s/he desires. The purpose of a Document Type Definition is to define the building blocks of an XML document. It defines the document structure with a list of allowed elements. The same holds for XML Schema – it only defines structure, though with a richer language.

XML is a mark-up language much like HTML. The former was designed to describe data and to focus on what data is and the latter was designed to display data and to focus on how data looks. HTML is about displaying information, whereas XML is about describing information. XML was created to structure, store and share information. The XML standard lets everyone create her/his own tags that annotate Web pages or sections of text on a page. Programs can make use of these tags in sophisticated ways, but the programmer has to know what the page writer uses each tag for.

In short, XML allows users to add arbitrary structure to their documents but says nothing about what the structures mean. However, tag-names do not provide semantics and the nesting of tags does not have standard meaning. Moreover collaboration and exchange are supported if there is underlying shared understanding of the vocabulary. Thus, XML is well-suited for close collaboration, where domain- or community-based vocabularies are used.

2.2.3 RDF/S

A key idea of XML was the separation of presentation from structure. With RDF a next step is taken by separating semantics from structure. This would allow using common semantic descriptions for different structural representations. RDF [23] stands for Resource Description Framework and its purpose is to describe resources on the Web. RDF is designed to be interpreted by computers. The basic RDF data model consists of three fundamental concepts: Resources, Properties and Statements.

Resources are the central concept of RDF and are used to describe individual objects of any kind, for example Web pages, people, hotels, books etc. Every resource has a URI, a Universal Resource Identifier, which can be a Web address or some other kind of unique identifier. Properties express specific aspects, characteristics, attributes, or relations between resources. For example, properties might be the number of rooms in a hotel, proximity to the beach etc. Finally statements are composed of a specific resource, together with a named property and the value of that property for that resource.

RDFS [24] is an abstract data model that defines relationships between entities (resources in RDF). RDF, in combination with RDFS, offers modeling primitives that can be extended according to the needs at hand. As a companion standard to RDF, the schema language RDFS is more important with respect to ontological modeling of domains. RDFS offers a more expressive vocabulary defined on top of RDF to allow the modeling of object models with cleanly defined semantics. The terms introduced in RDFS build the groundwork for the extensions of RDFS.

Finally RDF has an XML-based syntax to support syntactic interoperability. XML and RDF complement each other since RDF supports semantic interoperability. RDF has a decentralized philosophy that allows incremental building of knowledge and its sharing and reuse. However, RDF Schema is quite primitive as a modeling

language for the Web. Some desirable modeling primitives are missing. For this reason we need an ontology layer on top of RDF/RDFS and consequently a standard like OWL.

2.2.4 OWL

OWL [25] is a language currently being standardized by the World Wide Web Consortium for defining Web ontologies and their associated knowledge bases. In OWL, an ontology is a set of definitions of classes and properties, and constraints on the way those classes and properties can be employed. An OWL ontology may include the following elements: taxonomic relations between classes, datatype properties (descriptions of attributes of elements of classes), object properties (descriptions of relations between elements of classes), instances of classes and instances of properties.

OWL is a set of three, increasingly complex languages: OWL Lite, designed to satisfy users primarily needing a classification hierarchy and simple constraint features; OWL DL, which includes the complete OWL vocabulary interpreted under a number of simple constraints (DL stands for Description Logics); and OWL Full, which includes the complete OWL vocabulary, interpreted more broadly than in OWL DL.

Finally, OWL deals with some issues that RDF cannot express: disjointness of classes, boolean combinations of classes, cardinality restrictions and local scope of properties.

2.3 Query Routing in P2P Networks

2.3.1 The Problem of Query Routing

Query routing in a peer-to-peer network is the process by which the query is routed to a number of relevant peers and consequently it is not broadcasted on the whole network. The problem of query routing concerns the discovery of relevant peers to the query after we have denoted which peers are considered as relevant. Thus, we first have to define the criteria that make us to decide whether a peer is relevant or

not. For example in some P2P systems relevant peers are these ones that match exactly all the query predicates. Secondly, we have to define the strategy on which routing will be based (e.g. based on routing indices) and all the required routing steps.

Surely in peer-to-peer systems the network topology and the category of P2P determine to a large extent the applied routing strategy. Hence, before describing a routing algorithm we have to look at the characteristics of the peer-to-peer network that it will be applied to. An efficient query routing aims for limiting consuming network bandwidth by reducing messages across the network and reducing total query processing cost by minimizing the number of peers that contribute to the query's results. Finally routing in P2P networks is crucial for the scalability of the network. In the next subsections we describe the dominant approaches at query routing and their applied peer-to-peer environment.

2.3.2 Query Routing Based on Routing Indices and Super-Peer network

Wolfgang Nejdl et. al in [26, 27, 28] presented the routing approach based on routing indices. This approach has been suggested and adapted under various scenarios. It is built upon an RDF-based peer-to-peer network. Queries and answers to queries are represented using RDF metadata which we can use together with the RDF metadata describing the content of peers to build explicit routing indices which facilitate more sophisticated routing approaches. Queries can then be distributed relying on these routing indices, which contain metadata information plus appropriate pointers to other (neighboring) peers indicating the direction where specific metadata (schemas) are used. These routing indices do not rely on a single schema but can contain information about arbitrary schemas used in the network.

The authors in this approach of rounding indices assume super-peer topology for these RDF Schema-Based networks, where each peer connects to one super peer only. Super-peers then connect to other super-peers and build up the backbone of the super-peer network. Super peers are arranged in the HyperCup topology [29]. With HyperCup O(log(N)) [N: total number of nodes] messages are sent in order to integrate the new super-peer and maintain a hypercube-like topology. Furthermore, for broadcasts, each node can be thought of as the root of a specific spanning tree through the P2P network. There are two kinds of indices that contribute to the

specified routing strategy: *super-peer / peer routing indices* (SP/P, see Figure 2 below) and *super-peer / super-peer routing indices* (SP/SP see Figure 3 below)

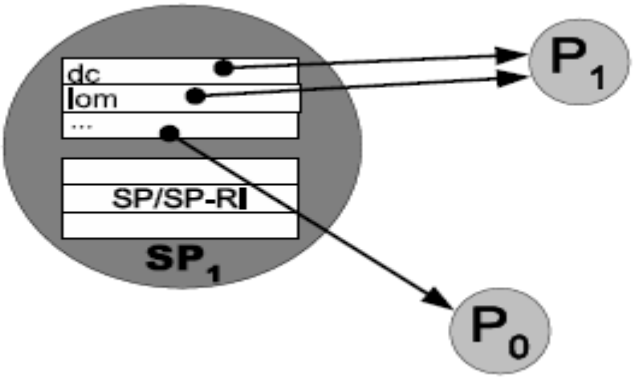

Figure 2: Super-peer/peer routing index

The first level index, the SP/P index, is an index which describes the characteristics of all peers connected to a specific super-peer, and thus guides the forwarding of queries from a super-peer to a connected peer. Thus, the super-peer/peer routing indices are used to forward the query to the respective peers only. These indices can contain the information about other peers or super-peers at different granularities: *schema identifiers, schema properties, property value ranges, individual property values.* These granularities are analyzed below:

- **Schema Index**. At the schema level it is assumed that different peers will support different schemas. These schemas are uniquely identified by their respective namespace; therefore the SP/P routing index contains the schema identifier and the peers supporting the respective schema.
- **Property/Sets of Properties Index**. Routing indices also contain properties or sets thereof thus enabling peers to support only parts of schemas. The properties are uniquely identified by namespace/ schema ID and property name and form the routing index entry together with those peer IDs where the properties are used.
- **Property Value Range Index.** For properties which contain values from a predefined hierarchical vocabulary an index which specifies taxonomies or part of a taxonomy for properties (the property value range) can be used. This is a common case in Edutella [21], because in the context of the

Semantic Web quite a few applications use standard vocabularies or ontologies.

- **Property Value Index.** For some properties it may also be advantageous to create value indices to reduce network traffic. This case is identical to a classical database index with the exception that the index entries do not refer to the resource, but the peer providing it. The index contains only properties that are used very often compared to the rest of the data stored at the peers. This would be used e.g. for string valued properties such as dc:language or lom:context.

Furthermore, they form the basis of the second level of indices, the SP/SP indices, which are derived from the SP/P indices, and facilitate routing within the super-peer backbone. Therefore queries are forwarded to super-peer neighbours based on the SP/SP indices, and sent to connected peers based on the SP/P indices.

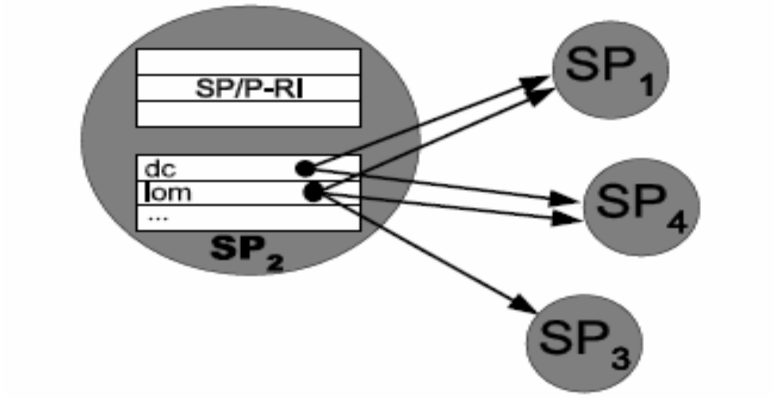

Figure 3: Super-peer/super-peer routing index

At this point we have to mention some assumptions and considerations that are made by the authors and are related to the updates of routing indices. An update of the SP/P index of a given super-peer occurs, when a peer leaves the super-peer, a new peer registers, or the metadata information of a registered peer changes (e.g., new attributes are added or deleted). The authors of this approach assume that each SP/P modification triggers the update process for SP/SP indices, though we can also collect the modifications for a given period and only then trigger the SP/SP update process. They further assume that the super-peers cluster peers according to their schema characteristics, so that peers connected to a super-peer usually have similar characteristics, and SP/P modifications trigger SP/SP index updates less frequently.

In order to view an example of this routing approach let's suppose the following sample query: Find any resource where the property dc:subject is equal to dc:language is equal to "de", ccs:softwareengineering and lom:context is equal to "undergrad". (dc, ccs, lom are namespaces of the corresponding schemas). The next figure (Figure 4) shows how peer Po sends the sample query mentioned above to its super-peer SP_1

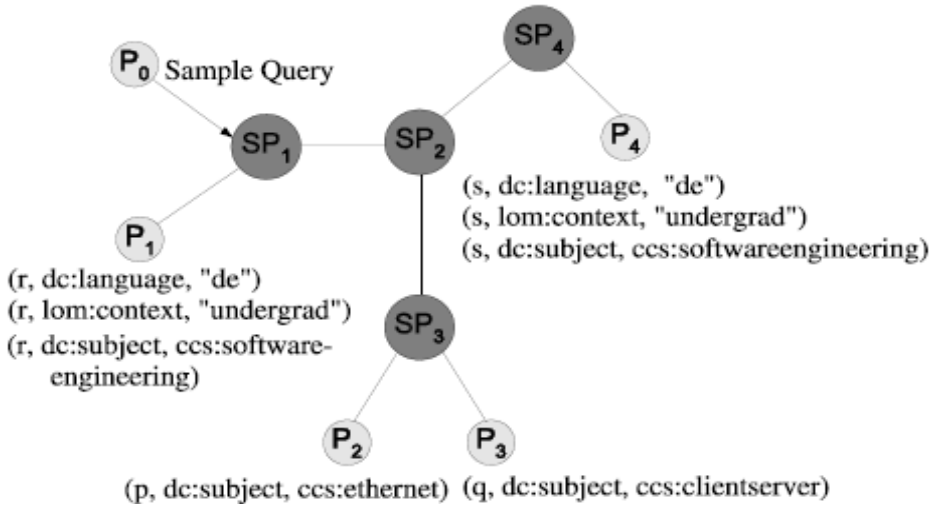

Figure 4: A routing example based on Routing Indices

In this example, the query could be answered by the peers P_1 and P_4, attached to SP_1 and SP_4 respectively. These contain metadata about resources r and s which match the query. Based on a schema-level-index, super-peer SP_1 forwards the sample query only to peer P_1 which supports the schemas lom and dc. Based on the property-level-index, the sample query in Figure 4 will be forwarded by SP_1 to P_1 because it is the only peer at SP_1 that using at least dc:subject, dc:language and lom:context. Similarly, the query is routed to P_4 by SP_4. At the specific example Figure 5 shows the SP/SP index of super-peer SP_2 at different granularities.

Granularity	Index of SP_2		
Schema	dc lom		SP_1, SP_3, SP_4 SP_1, SP_4
Property	dc:subject dc:language lom:context		SP_1, SP_3, SP_4 SP_1, SP_4 SP_1, SP_4
Property Value Range	dc:subject dc:subject	ccs:networks ccs:software-engineering	SP_3 SP_1, SP_4
Property Value	lom:context dc:language	"undergrad" "de"	SP_1, SP_4 SP_1, SP_4

Figure 5: SP/SP index of SP_2 at different granularities

It is obvious that indices only help if they can exploit and express regularities present in the peer and data distribution. Clustering peers therefore is a necessary ingredient for improving index effectiveness and thus query efficiency. At this basic approach clustering is based on the idea of integrating peers into locations already populated with peers of similar characteristics. Specifically the authors suggest the use of HyperCup partitions [29] and the use of frequency counting algorithms [30]. However these clustering methods are simple and not flexible enough. For this reason, a group of scientists and researchers including Wolfgang Nejdl suggest in [31, 32] a more advance technique for the assignment of peers to Super-Peers and their applied clustering. By this way they suggest an alternative approach based on Semantic Overlay Clusters.

2.3.3 Query Routing Based on SOCs and Information Peer Models on Super-Peer networks

The advanced technique of [31, 32] is also applied for Super-Peer Schema-Based peer-to-peer networks. Based on predefined policies a fully decentralized broadcast and matching approach distributes the peers automatically to super-peers. The basic idea here is that the super-peer establishes and maintains a specific *Semantic Overlay Cluster* (SOC). SOCs define peer clusters according to the metadata description of peers and their contents. Similar to the creation of views in database systems Semantic Overlay Clusters are defined by human experts. They act as virtual, abstract, independent views of selected peers in a Schema-Based P2P system.

The sum of all definitions regarding one SOC called as *SOC policy* (or policy for short). The policy states the conditions on which a peer is able to join a SOC. To state the policy the authors rely on a notation inspired by Event-Condition-Action (ECA) rules in active databases which is enhanced with logical operators. Thus rules have the following form: *ON event IF condition DO action.* Table 1 presents some rules within a clustering policy.

No	Event	Condition	Action	Explanation
1.1	$P_{Enter}(p,c)$	constraints == True	$Action = Approve(p,c)$	Peer p approved at Cluster c
1.2	$P_{Enter}(p,c)$	constraints == False	$Action = Reject(p,c)$	Peer p rejected from Cluster c
2.1	$P_{Leave}(p,c)$	-	$Action = Delete(p,c)$	Peer p deleted from Cluster c
3.1	$P_{Check}(p,c)$	constraints == True	$Action = Approve(p,c)$	Peer p (re-)approved at Cluster c
3.2	$P_{Check}(p,c)$	constraints == False	$Action = Reject(p,c)$	Peer p rejected from Cluster c

Table 1: Rules within a clustering policy

At this example clustering policy is based on some RDF properties (usesSchema, classifiedBy, taxonPath):

ON Enter (Peer p, Cluster c)
IF (
(usesSchema="http://purl.org/dc/elements/1.1/")
AND (classifiedBy="http://swebok.org")
AND (taxonPath >= "http://swebok.org/SoftwareDesign")
) DO Approve(Peer p, Cluster c)

By relying on an already established logical language, like Datalog, the P2P network supports the automated identification of suitable peers for a SOCs within a given search space of dimensions.

Similar to the definition for semantic overlay networks [10], the authors in [32] assume existing information provider peers and existing super-peers as nodes in a physical network. Then a semantic overlay cluster is defined as a link structure within a physical network (N) given a set of links from information provider (p) to a particular super-peer (s). Each SOC_L supports at least 2 functions: *Join(p_i, L)*, where links (p_i, s_j , L)) between a super-peer and a information provider peer are created and *Leave(p_i, L)* where they are dropped. Figure 6 shows all the basic concepts of this approach.

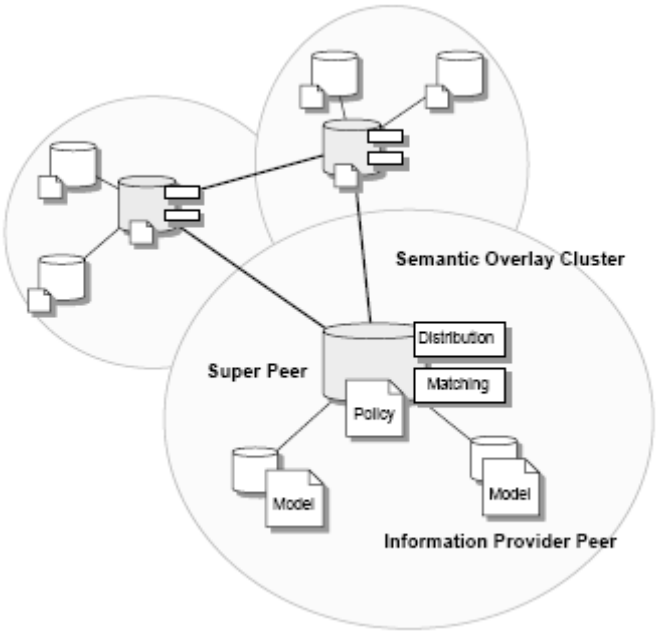

Figure 6: Super-Peer Network with Clustering Policy Information Provider Model

The authors focus on the realization of the join function. Thus, they consider that requests for a join are made by issuing a meta-data based model m_i of a particular p_i to the network. Also they assume that every information provider provides such a model. For joining the network an information provider peer chooses an arbitrary super-peer in the network and forwards its model to the super-peer. Since each cluster is related to one super-peer s_j and expresses explicitly its demand for information provider peers by a clustering policy c_j, the authors model a match between a clustering policy c_j and an the model of an information provider m_i as a function *Match* (m_i, c_j). Generally, matches can either be *exhaustive, partial, fuzzy or ontology-based*.

In the case of *exact match* an information provider peer only joins a super-peer when its model matches exact with the clustering policy. The information provider peer may also join the super-peer if only some attributes of the model match with the clustering policy. The last is the case of *partial* match whereas if similar attributes of the model match with the clustering policy then the match is called *similar.* The more sophisticated case includes collection and matching of attributes which are part of an ontology. It is the case of *ontology-based* match.

The matching process between clustering policies and information provider peers models operates in two stages. Firstly, we have the matching of the information

provider peer model with each local super-peer specific clustering policy according to the local implemented matching engine. Secondly, the information provider peer model is broadcasted within the whole super-peer network to all super-peers. Figure 7 shows schematically the matching process.

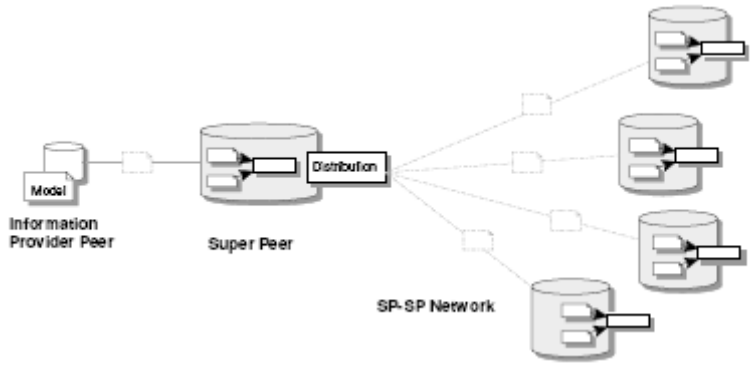

Figure 7: Matching and distribution of models in the Super-Peer Network

2.3.4 Routing Indices and/or SOC Approach

These two approaches are not contradictory. The second one (SOC Approach) in some ways fills in the first one (Routing Indices Approach). We believe that they easily can be combined to provide a more sophisticated routing technique. The approach of routing indices employs a weak clustering strategy and needs to accommodate index updates. In the approach of semantic overlay clusters we have to be careful at the matching process of the information provider peer model which it depends upon our clustering policy. For the clustering policy we suggest that we could use the information taken from routing indices at specified granularities. For example one rule of the potential clustering policy could be the match of namespaces at the schema granularity. Our general conclusion from these two relative approaches is at first that the Super-Peer network topology seems to be the most suitable for Schema-Based peer-to-peer networks since it can support heterogeneous Schema-Based systems with different metadata schemas and ontologies. The last is crucial for the Semantic Web. Also, as long as super-peers are getting the messages and they are not broadcast into the network, we have an efficient usage of network bandwidth by limiting the required transferred messages. In addition these two approaches fit well at this network topology, since they take advantage the role of super-peers. Finally we

should point out that different parts of these approaches have been used in the Edutella [20] project.

2.3.5 Query Routing Based on Social Metaphors and Shortcut Indexes

Another approach for query routing presented in [33] which is based on social metaphors. It defines a method for query routing called REMINDIN' (Routing Enabled by Memorizing INformation about DIstributed INformation). This routing method lets:

- peers observe which queries are successfully answered by others
- memorizes this observation
- subsequently uses this information in order to select peers to forward requests to.

Specifically, the basic steps of REMINDIN' routing method are the following:

- selects (at most) two peers from a set of known peers based on a given triple query, hence avoids network flooding and memorizes this observation
- forwards the query to them
- assesses and retains knowledge about which peer has answered which queries successfully.

In contrast to [26, 27, 28] this is a lazy learning approach [34] that does not advertise peer capabilities upfront, but that estimates it from observation. The main advantage of this approach is the self organization of P2P system. In other words, REMINDIN' supports query routing capability that mimics what a person is doing in a social network:

- she retains meta-information about what other peers know
- she might not even ask the others about their knowledge, but observe it from communication
- she does not have a fixed schema, but easily builds up new schematic or taxonomic knowledge structure
- she then decides to ask one or a few peers based on how she estimates their coverage and reliability of information about particular topics

From the above we can conclude that by this method we achive reduction of messages broadcasting, but if the knowledge at peers is limited the query's results and the effectiveness of this approach could become poor. In [35], the authors provide some extensions to the REMINDIN' technique by introducing new shortcut and ranking strategies of peers. By these additions peers can monitor:

- which other peers frequently respond successfully to their requests for information
- which peers ask similar questions
- which peers provide many documents or which peers have asked many questions to a broad range of topics in the past.

When a peer discovers such information, then it locally stores in a shortcut. Each shortcut represents an additional link on top of the default network layer of the peer-to-peer systems. Peers benefit from shortcuts by routing its queries directly to other peers along the shortcut overlay. Information from all shortcuts is eventually combined to decide to which peers a query will be sent. Shortcuts are created in an implicit manner to peers that have successfully answered queries in the past (Content Provider Layer) and peers that have asked similar queries in the past (Recommender Layer). It is assumed that peers can recommend relevant content providers, because of their previous efforts to get hold of such information. Also, another assumption is that peer continuously learns from new peers joining the network and "forgets" obsolete peers over time.

To further accelerate the learning process the Bootstrapping Layer is introduced. It contains peers that have established a high level of knowledge about other peers in the network. These peers are fast and implicitly discovered by peers with none or only few local knowledge about other peers in the network and are used as initial starting point for document queries. When a new peer enters the network, it has not yet stored any specific shortcuts in its index. Then it joins in the Default Network Layer. Default network shortcuts connect each peer p to a set of other peers (p's neighbors) chosen at random, as in typical Gnutella-like networks.

The ranking of shortcuts defines the routing strategy, since according to the rank the query is routed to a set of peers. The rank of the content provider depends on the similarity between a query and a local stored query dependent shortcut. This

approach uses a similarity metric [36] for topic hierarchies, but the authors report that it can be applied for other similarity functions as well. Generally, each peer forwards the query according to the local shortcuts with the highest similarity to this specific query. Furthermore, the bootstrapping capability of peers is determined by the number of shortcuts the peer has created and the number of remote peers it knows.

All the above considerations and ideas contribute to the proposal of the INGA algorithm [37, 38] by the same authors. INGA is a novel p2p algorithm where each peer plays the role of a person in a social network. Facts are stored and managed locally on each peer constituting the 'topical knowledge' of the peer. A peer responds to a query by providing an answer matching the query or by forwarding the query to what he deems to be the most appropriate peers. For the purpose of determining the most appropriate peers, each peer maintains a *personal semantic shortcut index*. This index is created and maintained in a lazy manner, i.e. by analyzing the queries that are initiated by users of the p2p network and that happen to pass through the peer. The personal semantic index maintained at each peer reflects that a peer may play the following four different roles for the other peers in the network (in decreasing order of utility): content providers, recommenders, bootstrapping network, default network. Figure 8 presents the shortcut overlay and the corresponding roles of peers

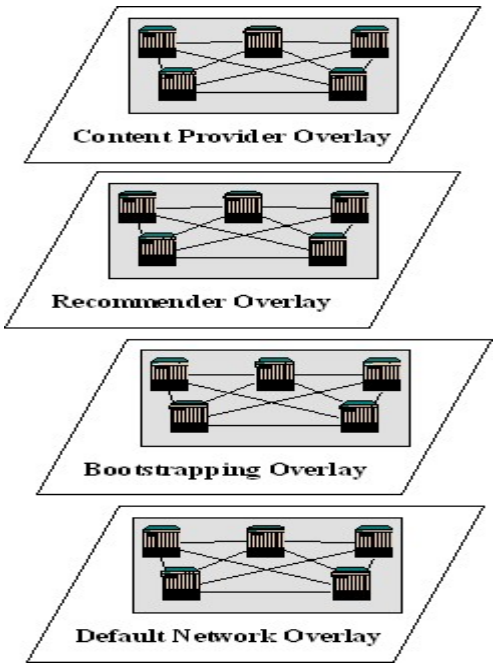

Figure 8: Shortcut overlay and roles of peers

The routing logic selects most suitable peers to forward a query to, for all own queries or queries forwarded from remote peers. The selection depends on the knowledge a peer has already acquired for the specific query and the similarity between the query and locally stored shortcuts (use of a similarity function [36]). Finally the authors present an Algorithm of Dynamic Shortcut Selection. The task of the INGA Dynamic Shortcut Selection algorithm is to determine best matching candidates to which a query should be forwarded. Relying on forwarding strategies and depending on the local knowledge for the topic of the query a peer has acquired yet in its index the main points of this algorithm are the following:

- forward a query via it's k best matching shortcuts.
- try to select content and recommender shortcuts before selecting bootstrapping and default network shortcuts.
- to avoid overfitting and accommodate a little volatility (especially in the form of new joining peers), queries are also randomly forwarded to some peers.

All this work constitute the approach of query routing based on social metaphors and shortcut indexes can be implemented on top of any unstructured Schema-Based peer-to-peer network. Part of the work is implemented within the SWAP [39] platform using RDF/S statements and SeRQL [40] query language. Also, similar techniques are used in [41], with the difference that the peers are using a shared ontology. One problem with these systems is that they are only document-based, namely they exchange documents across the p2p network. This occurs because, ranking of INGA peers is document-based (uses simple similarity measures such as TXDIF) assuming that each document belongs to a topic that corresponds to a term. In addition, the selection of peers is based on simple similarity measures e.g. matching just the query topic. The main limitation of this approach is the unavoidable flooding of the network with messages, when a new peer (has not yet stored any shortcuts) enters the network, or exploits lower levels of the INGA peer network.

2.3.6 Query Routing Based on Query Patterns

Recently, a different routing approach has been presented in [42]. As a part of the SQPeer Middleware it is presented a semantic query routing algorithm which is

based on the idea of Semantic Overlay Networks [10], RDF-Based peer-to-peer networks and RVL views as peer advertisements [43]. It can be build on any peer-to-peer architecture. As query language it uses the RQL [43]. Each peer node in SQPeer provides RDF descriptions that conform to a number of RDF schemas. Peer nodes with the same schema can be considered to belong to the same SON. In the upper part of Figure 9 below we can see an example of the schema graph of a specific namespace (i.e., n1) with four classes, C1, C2, C3 and C4, that are connected with three properties, prop1, prop2 and prop3. There are also two subclasses, C5 and C6, of classes C1 and C2 respectively, which are related with the sub-property prop4 of the property prop1. Queries in SQPeer are formulated by client-peers in RQL, according to the RDF schemas they use to create their description bases or to define virtual views over their legacy (XML or relational) databases.

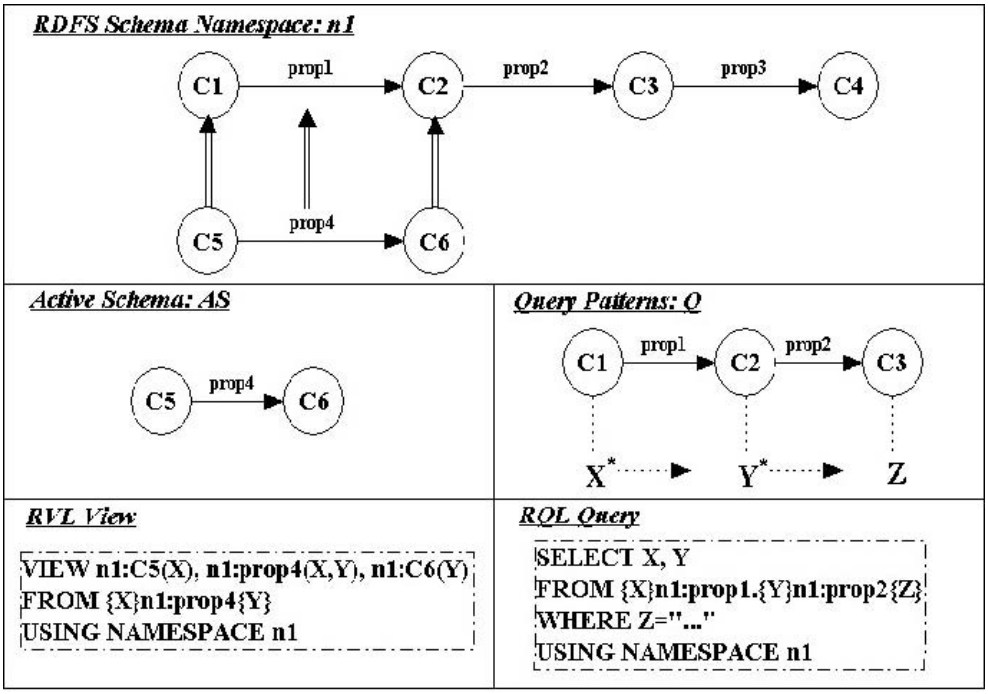

Figure 9: RDF/S schema namespace, peer active-schema and query pattern graph

In this context, there is a need to reason about query/view containment in order to guide query routing through the peer bases of the system. To this end, the authors introduce the notion of *query patterns* capturing the schema information employed by an RQL query. This information is mainly extracted from the path expressions appearing in the from clause. In the bottom right part of Figure 9 above it is shown an RQL query returning all the resources are bound by the variables X and

Y. In the from-clause, the employed path expressions imply a join on the Y resource variable between the target of the property prop1 and the origin of the property prop2. The where-clause filters the returned resources according to the value of variable Z. Filtering conditions are not taken into account by RQL query patterns. The right middle part of Figure 9 illustrates the query pattern graph of query Q, where X and Y resource variables are marked with "*" to denote projections.

Peer base advertisement in SQPeer relies on virtual or materialized RDF schema(s). Since these schemas contain numerous RDF classes and properties not necessarily populated with data in a peer base, we need a fine-grained notion of schema-based advertisements. The active-schema of a peer node is essentially a subset of the employed RDF schema(s) for which all RDF classes and properties are (in the materialized scenario) or can be (in the virtual view scenario) populated. The active-schema may be broadcast to (or requested by) other peer nodes, thus informing the rest of the P2P system of what is actually available inside the peer bases. The bottom left part of Figure 9 above, illustrates the RVL statement of a peer active-schema. This statement "populates" the classes C5 and C6 and the property prop4 (in the view-clause) with appropriate instances from the peer's base (in the from-clause). In the middle left part of this figure it is shown the corresponding active-schema graph obtained by this view.

At this approach query routing is responsible for finding the relevant to a query peers by taking into account data distribution (vertical, horizontal and mixed) of peer bases committing to a SON RDF/S schema. The query/view subsumption techniques of [45] are employed to determine which part of a query can be answered by an active-schema and rewrite accordingly the query sent to a peer. The SQPeer query routing algorithm takes as input a query graph and annotates each involved path pattern with the peers that can actually answer it, thus outputting an annotated query graph. A pseudocode description on how this algorithm works is given at Figure 10 below.

Query-Routing Algorithm:

1. A peer P receives an RQL query Q.
2. Peer P parses the query Q and creates the corresponding query pattern graph by obtaining the involved paths.
3. For each pattern, the matching algorithm is performed.
   (a) Compare the path pattern with all known active-schemas.
   (b) If the active-schema graph is subsumed by the selected path pattern, then it is annotated with the name of the peer owning the active-schema.
4. Output annotated query graph.

Figure 10: SQPeer Query Routing Algorithm

In general, SQPeer provides a good routing technique which utilizes the notions of Semantic Overlay Networks and RVL views upon a RDF-Based P2P network. However since each view (active-schema) corresponds to a peer advertisement, it should be broadcasted in such a way to inform the whole peer-to-peer network. This process consumes enough network bandwidth, as long as peers leave or join frequently the P2P network. In this case a suitable peer-to-peer architecture is needed in order to limit the required transferred messages in each instance of the algorithm. Furthermore, since SQPeer's routing algorithm uses the query/view subsumption techniques of [45], constrainedly each view should be populated with data from a relational or an XML peer base.

2.4 Query Processing in P2P Networks

2.4.1 The Problem of Query Processing

Query processing is the next step after the query routing. Namely, after the query is routed to a set of appropriate peers, query processing undertakes to combine the results from each peer and to return the final ones to the peer that makes the original query i.e. user of the P2P system. Generally query processing is dependent on the query routing strategy. A nice performance can be achieved if both of them can be cooperated together in a smart way. Good performance in a peer-to-peer network entails fast retrieval of data without large bandwidth consumption as soon as the processing steps are executed in a distributed way.

Query processing in peer-to-peer network is a multidimensional topic that many authors have worked on it suggesting a variety of techniques according to the hypotheses and the aspect of the problem each author formulates. But, for Schema-Based peer to peer networks there is a minimal work in query processing of such systems that is based on query planning and optimization. The new trend in query processing is the adaptation of top-k retrieval algorithms in order to get back the results quickly and without any large processing cost. This technique has just started to apply for distributed environments. However because the idea of top-k queries has been first applied in relational databases [14], there are some open issues that have to be defined for each applied distributed scenario.

2.4.2 Query Processing based on Query Planning and Optimization

The authors of routing indices routing approach [27, 28, 29] has done some work on query processing based on Query Planning and Optimization for Super-Peer Schema-Based P2P networks. They try to combine the advantages of their routing approach with their introduced query processing technique [12, 13]. Therefore, in contrast to traditional distributed query optimization, the plan is not generated statically at one single host. In their approach, super-peers generate partial query plans which are executed locally and the remainders of the query are pushed to the neighbours. Thereby, plan generation involves five major steps as depicted in Figure 11 below:

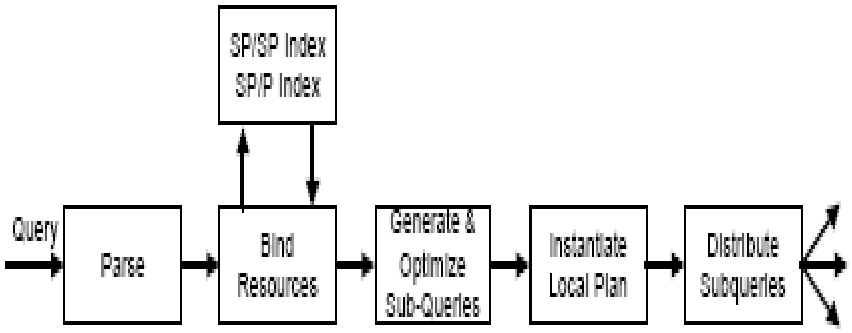

Figure 11: Query Planning Generation at Super-Peers

Thus firstly the incoming query is parsed, secondly recourses are bound based on index information and thirdly the subqueries are generated based on bindings. After that, the local query is instantiated at super-peers and the last step is the

distribution of subqueries to neighbouring (super) peers. The authors of this processing approach suggest that it can be implemented as part of their QueryFlow [46, 47] system or Edutella [20].

A new technique applied for SQPeer Middleware [42]. Query processing in SQPeer takes the responsibility of generating distributed query plans according to the information returned by the SQPeer routing algorithm. Therefore, the creation of the query plans is based on an annotated query pattern, which in turn is formulated by considering routing information (relevant peer views) gathered during the routing phase. The produced query plan specifies precisely how the query is going to be deployed and executed at the selected peers contributing to the final answer. Specifically, initially, the query is parsed and a query pattern is handled by the routing phase. A fragmentor is responsible for breaking the query into distinguished fragments and for each one the lookup service is utilized to find relevant routing information. Then, a data localization algorithm produces an annotated query pattern by annotating each fragment of the query with the peers that can handle it. The produced pattern is then sending to the query planning algorithm, where an appropriate query plan is produced by translating the pattern into the SQPeer query algebra. Since this query plan contains no optimizations, it is passed to an optimizer, who undertakes the physical optimization process.

The optimizer applies heuristic and/or cost-based techniques producing an optimized query plan taking into account inter and intra-peer query processing and communication cost. Finally, the optimized plan is sent to the execution engine responsible for forwarding the already distinguished subplans to the appropriate peers and monitoring their evaluation. Peer communication is achieved by the use of appropriate communication channels that additionally provide the means for query plan adaptation during query execution in case of run-time failures.

Eventually, another approach has been introduced in [48]. Their strategy is based on so called mutant query plans which encapsulate partially evaluated query plans and data. In this approach, loss of pipelining during execution limits the general applicability for distributed query processing. We will not talk further about this way of processing, since at this thesis we are dealing with top-k processing in peer-to-peer networks. An introduction of top-k queries and the related work at our focused domain follows.

2.4.3 Query Processing for top-k queries

2.4.3.1 Top-k queries in general and P2P networks

The idea of top-k queries was first introduced in [14] and applied for relational databases. Ronald Fagin presented in [14] the Fagin Algorithm (FA) to solve the ranking aggregation problem for multimedia database systems. Simply top-k queries return only the k best results according to a given criterion. Generally, top-k queries on multidimensional datasets compute the k most relevant or interesting results to a partial-match query, based on similarity scores of attribute values with regard to elementary query conditions and a score aggregation function such as weighted summation. The state of the art on top-k queries for middleware applications has been defined by the seminal work on the Threshold Algorithm (TA) in [49] also by Fagin et al.

After that, several approaches and efficient strategies for top-k query processing have been developed concerning classical Relational Database Management Systems (RDBMS) such as [50, 51, 52]. Algorithms for top-k retrieval in databases generally try to minimize the number of database objects that have to be accessed before being able to return a correct result set of the k best matching objects. These algorithms (rank/score-based) are firstly applied to all objects and to all query predicates, and secondly aggregate of score values to get the best results back. For RDBMS the matching objects correspond to matching tuples.

In the field of peer-to-peer networks only very few authors have written about supporting top-k retrieval algorithms in such distributed environment. Despite this fact top-k query processing provides undoubtedly a good technique since the P2P users are at most cases interested in a few most relevant answers to their query which are returned by the top-k algorithm. But, except of the query model that has to be defined well in all cases, for the case of p2p we have to define what kind of objects the peers are own, the resulting assumptions and how the communication cost of the applied top-k algorithm would keep in low values the total cost. In large distributed environments like peer-to-peer systems the fewer messages are transmitted across the network (in this case due to the execution of top-k processing algorithm) the more scalable and efficient are the specified systems. Surely, the processing cost

additionally depends on how distributed the top-k retrieval algorithm is executed and how complicated procedures are needed in order to return the final top-k results.

2.4.3.2 Assumptions on existing approaches of top-k query processing in P2P networks

Approaches for top-k query processing in P2P networks has been studied only recently. It seems to be a new requirement in query processing for large distributed environments. Before we start an overview of the existing techniques, we mention in this subsection some common issues. All approaches assume that:

- each peer (source) has saved a sorted/unsorted list of pairs: (object_id, score)
- Object_id is unique (i.e. Object O1 in peer 1 is the same with Object O1 in peer 2).
- Each pair is related with one attribute (property).

Almost all approaches assume that these pairs pre-exist. There is only one approach [53] that deals with the ranking and the use of ranking methods (Topic-distances in Taxonomies, TFxIDF). Finally, all these top-k query processing approaches support only of selection queries. There is minimal work to support join queries even in relational databases [54, 55, 56]. We classify the existing techniques in top-k processing for peer-to-peer networks into four categories and we present them at the next subsections.

2.4.3.3 The Probabilistic – Histograms Approach

In this category of top-k query processing in peer-to-peer networks belongs a family of algorithms that are introduced by independent research groups and computer scientists [57, 58, 59, 60]. The basic idea of these approaches is that given a top-k query then we can provide a probabilistic guarantee that x percent of the retrieved objects are among the top-k objects which we would get if we had asked all peers in the system. In order to be able to prune away objects there is a need of information about data distribution at each peer. The final pruning of objects under specific probabilistic guarantees is achieved using data structures like routing filters and histograms. Routing filters collect information about all attributes with high

frequencies. Also they can combine information about schema and instance level. What is important for optimizing the evaluation of top-k queries is the approximation on instance level provided by the histograms. Histograms approximate data distributions by partitioning a sort parameter into intervals (buckets). An approximated source parameter value is stored for each bucket.

In KLEE which is presented in [58] a more sophisticated but similar technique is used. Each peer maintains a set of statistical metadata describing its index list. In particular, histogram-based information is maintained to describe the distribution of scores in the index list. For simplicity, the authors assume that peer histograms are equi-width, consisting of n cells, each cell being responsible for (1/n)th of the score range. Associated with each cell i, each peer maintains lower, upper and average values plus frequencies between these bounds. Furthermore, KLEE uses Bloom filters [54] to compactly represent, for each histogram cell, the set of documents (since it is document-based framework) whose scores fall in this cell. This information, coupled with the statistical metadata contributes to the basic steps of the top-k approximation retrieval algorithm.

The proposed solutions that come from the probabilistic – histograms approach fall back to broadcast when the desired number of results is too high or when the user asks for a good degree of accuracy. As we know this is an important disadvantage since these solutions are applied for peer-to-peer systems. Furthermore the authors assume that each participating peer already owns histograms for all of its neighbors and the queried attributes. In addition in most cases they use one-dimensional histograms which means that the ranking functions defined over only one attribute. Also, approximate top-k algorithms that based on Threshold Algorithm (TA) need several round-trips in order to retrieve the final results. The last occurs because TA does not take into account data distribution and it works until it finds the k objects whose aggregated scores are no less than the current suitable threshold. Finally we could conclude that there is a general trade-off between result quality and expected performance.

2.4.3.4 The Nejdl et. al Approach

Wolfgang Nejdl et al in [53] tried to combine his ideas of semantic query routing based on indices [27, 28, 29] and propose a decentralized top-k query

evaluation algorithm for peer-to-peer networks which makes use of local rankings, rank merging and optimized routing based on peer ranks, that promises to minimize both answer set size and network traffic among peers. All the approach relying on super-peer backbone organized in the HyperCup [29] topology upon a RDF-Based P2P network. This top-k answering and routing algorithm is based on dynamically collected statistics that put them in local indexes. The steps of this algorithm are in short the following:

- Each peer computes local rankings for a given query, results are merged and ranked again at the super-peers and routed back to the query originator.
- On the way back, each involved super-peer again merges results from local peers and from neighbouring super-peers and forwards only the best results, until the aggregated top-k results reach the peer that issued the corresponding query.
- While results are routed through the super-peers, they maintain statistics (on local indexes) which peers / super-peers returned the best results.
- This information of local indexes is subsequently used to directly route queries that were answered before mainly to those peers able to provide top answers.
- Additionally, a small percentage of queries will additionally be forwarded randomly to enable lazy update of these indices to adapt to changes in the peer-to-peer network.

This is the only approach that deals with the ranking of resources, since all the others assume that the ranking has been done before the top-k processing algorithm starts. However it uses simple similarity measures good for document-based systems such as TFxIDF [61] or topic distances in taxonomies which is useful if we use a shared ontology.

Upon all these ideas Wolfgang Nejdl and Wolf-Tilo-Balke introduced a little more sophisticated top-k processing framework in [62]. But the basic limitations of this approach have remained. Thus, the first time, all peers have to participate in processing the query while several round-trips are required in order to retrieve the final result. This often leads to situations where peers have to wait for each other.

Also, in the case where the query is not contained in indexes the algorithm becomes time and network consuming. Eventually, this work concentrates on a very simple query language, and it's applicability to more complex languages is unclear. Comparing with the other approaches this one does not aggregate scores from all peers, because it is based mainly on its local indexes to decide for top-k matching objects.

2.4.3.5 The Marian et. al Approach

Amelie Marian et. al. first in [63] introduces the Upper Algorithm, an algorithm for evaluating top-k queries over web-accessible databases. Its applied query scenario is related to a (centralized) multimedia query scenario where attributes are reached through several independent multimedia "subsystems," each producing scores that are combined to compute a top-k query answer. If we consider that each "subsystem" which is a source in the Upper represents a peer of a peer-to-peer network, then surely this technique can be used for distributed environments. Three types of sources are used at this approach based on their access interface: random (R-Source), sorted (S-Source) and both random and sorted (RS-Source). The Upper algorithm requires one SR-Source and any number of R-Sources. Upper allows for more flexible probe schedules in which sorted and random accesses can be interleaved even when some objects have only been partially probed.

The Upper Strategy selects a pair (object, source) to probe next based on the property*: the object with the highest upper bound will be probed before top-k solution is reached.* Specifically the steps of the Upper Algorithm are the following:

- Choose object with highest upper bound.
- If some unseen object can have higher upper bound:
  - Access S-Source S
    Else:
  - Access best R-Source Ri for chosen object
- Keep best *k* objects
- If top-*k* objects have final values higher than maximum possible value of any other object, return top-*k* objects.

In [64] Amelie Marian et al. presented pUpper, an improved version of Upper that tries to maximize source-access parallelism to minimize query response time, while observing source-access constraints. pUpper allows for any number of SR-Sources and R-Sources. The idea of pUpper strategy is that it precomputes a list of objects to access per source, based on expected score values. Also, pUpper associates a queue with each source for random access scheduling. These queues are regularly updated by calls to a method named *GenerateQueues*. Figure 12 below depicts the pUpper algorithm:

Algorithm *Upper* (Input: top-k query q)
(01) Initialize $U_{unseen} = 1$, $Candidates = \emptyset$, and $returned = 0$.
(02) While ($returned < k$)
(03) If $Candidates \neq \emptyset$, pick $t_H \in Candidates$ such that $U(t_H) = \max_{t \in Candidates} U(t)$.
(04) Else t_H is undefined.
(05) If t_H is undefined or $U(t_H) < U_{unseen}$ (unseen objects might have larger scores than all candidates):
(06) Use a round-robin policy to choose the next *SR-Source* D_i ($1 \leq i \leq n_{sr}$) to access via a sorted access.
(07) Get the best unretrieved object t from D_i: $t \leftarrow \texttt{getNext}(D_i, q)$.
(08) Update $U_{unseen} = ScoreComb(s_i(1), \ldots, s_i(n_{sr}), \underbrace{1, \ldots, 1}_{n_r \text{ times}})$,
where $s_i(j)$ is the last score seen under sorted access in D_j. (Initially, $s_i(j) = 1$.)
(09) Else If t_H is completely probed (t_H is one of the top-k objects):
(10) Return t_H with its score; remove t_H from *Candidates*.
(11) $returned = returned + 1$.
(12) Else:
(13) $D_i \leftarrow SelectBestSource(t_H, Candidates)$.
(14) Retrieve t_H's score for attribute A_i, s_i, via a random probe to D_i:
$s_i \leftarrow \texttt{getScore}(D_i, q, t)$.

Figure 12: The pUpper Algorithm

One of the significant advantages of the Marian et. al approach is that their top-k processing strategy doesn't require complete knowledge about the scores at each step. Therefore (p)Upper selects an object source pair to probe next, based on expected set of probes. Also, (p)Upper computes "best subset" (by using an appropriate method) of sources that is expected to compute the final score for k top objects and finally discard other objects as fast as possible (pruning of top-k candidate set). Furthermore, it is the first approach in distributed environment that talks in practice about combined scored based on more than one attribute.

However, maybe the main drawback of this strategy is that only one source can be accessed at a time. This is too restrictive if we consider a peer-to-peer network, where a large number of peers must wait for the others at each step of the algorithm to access the suitable (Object, Score) pairs. (p)Upper enables parallel top-k processing and emphasize only on reducing query response time through the use of queues in order to gain the lost time from the delay of accesses at each peer (source). But, in a

widely distributed scenario (p)Upper may incur in a potentially unbounded number communication (messages) rounds. The last, is a characteristic of TA-style algorithms and the number of rounds depends on data distribution. Finally, if parameter L which indicates the length of the random-access queues is not chosen correctly pUpper might perform "useless" probes.

2.4.3.6 Three Phase Threshold Approach

Another approach to top-k query processing comprises of algorithms that are consisted of three phases and use thresholds in order to finally return the top-k results. These algorithms are designed to answer top-k queries on large scale networks efficiently. Although they belong to Threshold Algorithm – Style (TA-Style), they overcome the problems of TA. The last cannot be applied to large scale networks because it works well only when the number of participating nodes m is small. When m is large the network traffic involved in the second round-trip can become excessive, regardless of choices of the block size. Also, an additional problem is that the latency of TA is unpredictable because the number of rounds varies by data input. For distributed networks, it's indubitably desirable to have an algorithm that terminates in a fixed number of round trips.

Before we start to present algorithms of this family we should introduce the problem formulation and some basic concepts. The authors assume that there are m nodes and one single central manager in a distributed system. Each node i is connected to the central manager and maintains a list of pairs (O, Si(O)), where O is an object and Si(O) is the score of the object. Also they assume that objects in each list are sorted in the descending order of their scores. If an object does not appear in the list of a node, its score in that list is zero by default. The central manager (central node) that initiates a top-k query and finally retrieves objects from the network with the k highest $f(S_1(O),...,S_m(O))$ where f is a monotonic function such as the sum function SUM to compute the overall score of an object. For simplicity, the authors assume the sum function but, in practice, this function could be a weighted sum to account for the relative importance of participating nodes.

The first work of this approach and the guide for the next ones is the *Three-Phase Uniform Threshold (TPUT)* was presented in [65]. To describe the Three-Phase Uniform-Threshold Algorithm (TPUT), we have to define the notion of partial sums

of objects which are calculated by the central manager. For an object O, the partial sum $S_{psum}(O) = S'_1(O) + \ldots + S'_m(O)$ where $S'_i(O) = S_i(O)$ if O has been reported by node i to the central manager, and $S'_i(O) = 0$ otherwise. The three phases of TPUT are the following:

Phase 1: Each node sends its top-k objects to the central manager. The central manager then calculates the partial sums for all objects seen so far and identifies the objects with the k highest partial sums.

Phase 2: Let τ1 be the partial sum of the kth object. This value is referred to as the "phase-1 bottom". The central manager first sends a threshold value T = τ1/m (m = number of nodes) to every node in the system. Then each node sends its objects to the central manager, whose scores are no less than T. The intuition is that if an object is not reported by any node, its sum must be less than τ1. Hence it cannot be a top-k object. Now the central manager can re-calculate the lower bound. It calculates the new partial sums for the objects seen so far. Then the new lower bound τ2 ("phase-2 bottom") is the partial sum of the kth object. An upper bound of each object's aggregated score is calculated by Usum(O) = S'1(O) + … + S'm(O) where S'i(O) = Si(O) if O has been reported by node i to the central manager, and S'i(O) = T otherwise. If the upper bound of an object's aggregate score is less than τ2, it can be pruned. After pruning, the set of objects left are the top-k object candidates.

Phase 3: This phase identifies the top-k objects. The central manager sends the top-k object candidate set to each node and each node in turn sends the scores of these objects to the central manager. Hence, the central manager can calculate the real scores for these objects and then identify the exact top-k objects.

TPUT reduces network bandwidth consumption by pruning away non-eligible objects based on their scores, and terminates in three round-trips regardless of data input. After TPUT has introduced, in [15] a group of scientists coming from University of California took TPUT as a base and presented three new algorithms called TPAT, TPOR and HT. These algorithms belong to the same category of Three Phase Threshold Approach. The *Three-Phase Adaptive-Threshold (TPAT)* algorithm

generalizes TPUT by exploiting data distributions using summary statistics to further enhance the pruning power of TPUT. TPAT (and HT as well) extends TPUT by relaxing the condition on how to divide the phase-1 bottom ($\tau 1$) among all nodes. By dividing $\tau 1$ to the number of nodes, this algorithm takes into account cases where some nodes may have larger score distributions than other with smaller distributions. The three phases of TPAT are the following:

Phase 1: same as TPUT

Phase 2: Instead of using a uniform threshold T $\tau 1/m$, the central manager divides $\tau 1$ non-uniformly into T_1 ... T_m according to some summary statistics sent from nodes. Then it sends T_1 ... T_m to node I ...node m respectively as their thresholds. The rest of Phase 2 is the same as TPUT except that the upper bound of each object's aggregated score calculated by $U_{sum}(O) = S'_1(O) + \ldots + S'_m(O)$ where $S'_i(O) = S_i(O)$ if O has been reported by node i to the central manager, and $S'_i(O) = T_i$ otherwise.

Phase 3: same as TPUT

The problem with TPAT is that generally it could be very expensive to use summary statistics to accurately estimate data distributions. For this reason the authors suggested *Three-Phase Object-Ranking (TPOR)* Algorithm. TPOR prunes non-eligible objects by their rankings (positions). By this way, it estimates data distributions, without a-priori knowledge. The three phases of TPOR are the following:

Phase 1: same as TPUT

Phase 2: The central manager broadcasts the list L of the top-k object IDs from the partial sum list to all the nodes in the network. Upon receiving the list L, for each object O_j in L, node i finds its local score $V_{i,j}$ (if O_j does not occur in the local list, $V_{i,j} = 0$) and determines the lowest local score T_i among all the k objects in L. Then node I sends the list of local objects whose values are >= T_i to

the central manager. Now the central manager calculates the partial sums of all the objects seen so far, and identifies the objects with the k highest partial sums. Let us call the kth highest partial sum "phase-2 bottom" and denote it by $\tau 2$. Then the central manager tries to prune away more objects. It calculates the upper bounds of the objects seen so far using $U_{sum}(O) = S'_1(O) + \ldots + S'_m(O)$ where $S'_i(O) = S_i(O)$ if O has been reported by node i to the central manager, and $S'_i(O) = T_i$ otherwise. Then the central manager removes any object O_j from the candidate set whose upper bound is less than $\tau 2$.

Phase 3: same as TPUT

From their evaluation the authors conclude that TPOR is more bandwidth-efficient than TPUT when handling the case that object rankings are similar across all nodes. Nevertheless, TPOR performs worse than TPUT in the case when object rankings widely vary across all nodes. To remedy such a situation, they proposed *Hybrid-Threshold algorithm (HT)*. This algorithm combines the advantages of both TPUT and TPOR, and as the evaluation proved it is very robust to different data distributions. HT has also the great advantage like TPOR of estimating data distributions without a-priori knowledge. All these characteristics make HT too attractive and competive to the other approaches.

In the next chapter of methodology we present the Hybrid-Threshold algorithm (HT) and adapt it into our peer-to-peer scenario. We also compare it with (p)Upper which seems to work well for distributed networks and is the only from all the above algorithms (that come from different approaches) that supports of estimating data distributions without a-priori knowledge to defend our choice of HT against all the others. Furthermore, we extend HT and introduce HT-p2p an improved version of HT that is adapted under peer-to-peer networks which are organized in a Super-Peer network topology.

2.5 Chapter Summary

In this chapter we described the main concepts that are used in our work, starting from the basic, such as peer-to-peer networks and semantic web. In addition

we denoted the problem of query routing and processing for distributed networks and we described in detail the dominant approaches for these two fundamental problems. Therefore we have a complete view on the state of the art work for the general problem of efficient usage and search in a P2P network.

Chapter 3

Methodology

In this chapter we present our methodology for efficient query routing and processing in peer-to-peer networks. The basic building blocks are the query routing strategy and the top-k query processing strategy that we suggest for an efficient framework designed for large scale distributed networks. This framework promises fast query answering, easy data sharing, stability, self organizing, autonomy, load-balancing, low bandwidth consumption across the applied p2p network.

3.1 Basic Context and directions

3.1.1 Formulation of the problem

At first we have to define the problem that we try to solve. Let's assume that we have a peer-to-peer network. Each peer has its own Knowledge Base (db). Each database we want to be self organized by each corresponding peer. We can view each peer as a person who shares his knowledge with other people (peers of the P2P network). It is desirable each person to have the ability to reuse others knowledge at least at database schema level. Our main goal is the easy sharing of knowledge bases which implies efficient exchange of data across the p2p network. In practise our goal would be achieved if each query is not broadcast into the whole network, but is routed to relevant peers. Going one step ahead, the efficiency and the good performance of the whole peer-to-peer network does not only depend on how the query is routed to relevant peers, but also on how it is processed by these relevant peers. Surely as we have also pointed out in the previous chapter the problems of query routing and processing is dependent of the applied distributed environment. Therefore, we first describe our applied p2p scenario and all our design decisions upon our suggested context.

3.1.2 Design Decisions for Peer-to-peer networks

To start with the category of peer-to-peer network that fits well with the above problem, we choose an *unstructured P2P environment*. The main reason for that is that we want peers to be free to manage their own data (self organization) since each peer has its own knowledge (data) base. Unstructured networks provide this functionality and additionally support of rich query languages upon these databases in contrary to structured DHT networks. Also, we suggest for our network topology a *Super-Peer network architecture.* Super-Peer networks combine the efficiency of a centralized search (super peers route to appropriate peers the query) with the autonomy, load balancing [66] and robustness to attacks provided by distributed search. As long as super-peers are getting the messages and they are not broadcasted into the network, we have an efficient usage of network bandwidth by limiting the required transferred messages. Furthermore by using a Super-Peer network topology we can take advantage of the heterogeneity of capabilities (e.g., bandwidth, processing power) across peers and can cluster them according to some defined criteria. Finally, Super-Peer based networks can provide better scalability than pure P2P networks and eliminate the phenomenon of bottlenecks which can potentially occur for broadcast-based networks (pure P2P).

Given our assumption that each peer is managing a database, our peer-to-peer network fits well with the idea of *Schema-Based P2P networks*. In addition our suggested Super-Peer topology can provide support for heterogeneous schema-based networks with different metadata schemas and ontologies. For this reason we can easily take advantage of *Semantic Web technologies* and by this way can build a peer-to-peer system that exchanges information across the network semantically.

In the Semantic Web, an important aspect for its overall design is the exchange of data among computer systems without the need of explicit consumer-producer relationships. *RDF* [23] and *RDF Schema* [24] are used to annotate resources on the Web thus providing the means by which computer systems can exchange and comprehend data. All resources are uniquely identifiable by an URI. The annotations about resources are based on various schemas that are built based on RDFS (and possible extensions) and are stored in what we call RDF repositories possibly using more than one schema. One important characteristic of RDF metadata is the ability to use distributed annotations for the same resource. In contrast to

traditional database systems, it is not necessary that all annotations of a resource are stored on one server. For example, one server might store metadata which include properties such as name for specific resources possibly using the Dublin Core metadata standard. Other servers also could hold metadata that provide properties for the same resources, possibly using other metadata standards / schemas. This ability for distributed allocation of metadata makes RDF suitable for the construction of distributed repositories like Schema-Based peer-to-peer networks.

Also, RDF schemas are flexible and extensible such that schemas can evolve over time, and RDF allows the easy extension of schemas with additional properties. As such RDF is capable of overcoming the problems of fixed and unchangeable metadata schemas which often occur in recent peer-to-peer (P2P) systems and shows the direction of knowledge reuse across the applied p2p network. Finally the functionality of RDF/S to define easily schemas attributes and ontologies, to extend them, to enrich them and to reuse them without any cost (i.e through the use of namespace mechanism and other RDF mechanisms) guide us to suggest a *RDF-Based peer-to-peer networ*k. To sum up our suggested peer-to-peer network has the following characteristics:

- It is unstructured
- It is Super-Peer based
- It is RDF/S Schema based

This suggestion about the P2P network provides us solutions to the basic formulation of the problem. Furthermore, we have taken into account of all these characteristics in our suggesting query routing and processing strategy in order to exploit its advantages and to adapt them under our completed proposed peer-to-peer framework. In the next sections we present our query routing and processing policy that is built upon an unstructured RDF/S Schema based peer-to-peer network which is organized upon a Super-Peer topology.

3.2 Query Routing Strategy

3.2.1 Basic features and query routing context

Ontologies are a key enabling technology for the Semantic Web. Their role is crucial for the development of large scale "semantic" information systems since they define formal semantics for information, consequently allowing information processing by a computer. In addition ontologies define real-world semantics, which makes it possible to link machine processable content with meaning for humans based on consensual terminologies. Thus, many systems that support semantic interoperability use at least an ontology that defines a specific domain and describe it. Ontologies can be built by using widely known semantic web standards such as XML [22] and RDF/S [23, 24] and specific languages like OWL [25]. Usually *ontologies* are deployed upon specified *taxonomies*. Taxonomies usually represent well defined relations in the Semantic Web.

We suggest that each database has its own *taxonomy of term*s that describe its contents (schema level). Therefore, each peer followed the model of [67, 68] and it can be thought of, as a simple source. A simple source consists of a taxonomy and an object base that indexes objects under the terms of the taxonomy. Terms are connected through *isA links*. In our case, each peer is a simple source which has the corresponding taxonomy which indexes the actual database that contains the real data (instance level). We also suggest that each peer has its own *RDF/S schema* which describes its database schema information and includes the corresponding taxonomy of terms that is related with the specific database and other user-defined relationships and properties. Furthermore we suppose that there are *two-way links* (cross links) between terms of different taxonomies. In practice each peer can make an *"in relation" isA link* by linking a term of its taxonomy with a term of another peer. This can be done very easily by using RDF/S mechanisms (i.e., namespace). In this way, each peer not only manages its specific schema and their indexing data, but it can enrich them by using other terms that are related to its own. See an example of two taxonomies of terms in Figure 13 below. It shows two taxonomies of terms, each one belongs to different peer and describes its database contents.

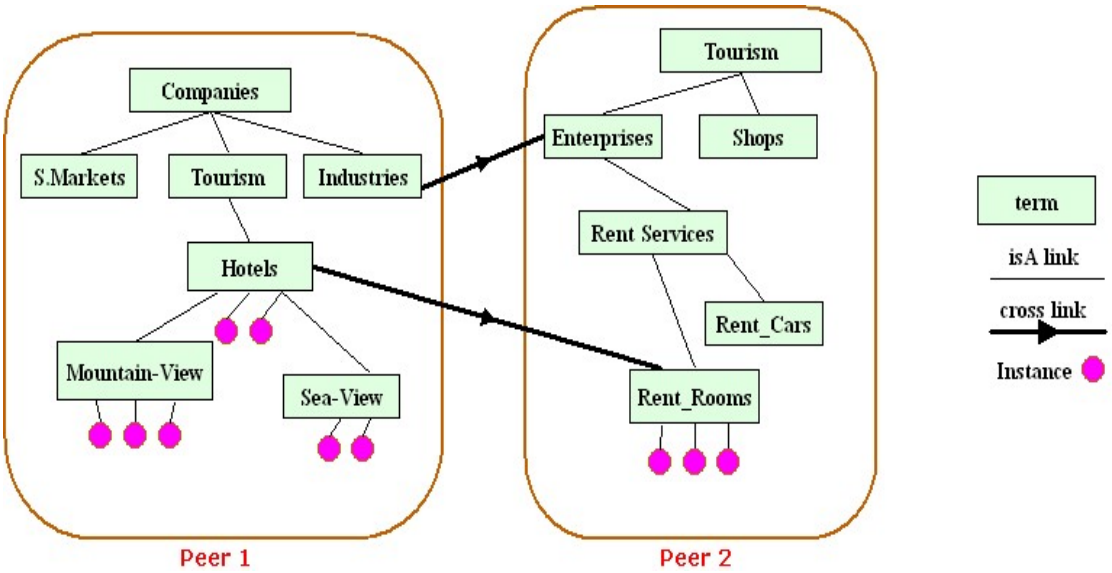

Figure 13: An example of two taxonomies

Undoubtedly for querying RDF/S schemas we need a query language with many capabilities and functionalities upon these schemas. *RQL* [43] is a typed language following a functional approach, which supports generalized path expressions featuring variables on both labels for nodes (classes) and edges (properties). It relies on a formal graph model that captures the RDF modelling primitives and permits the interpretation of superimposed resource descriptions by means of one or more schemas. The novelty of RQL lies in its ability to smoothly combine schema and data querying while exploiting the taxonomies of labels and multiple classifications of resources in a transparent way. Considering a set of well defined criteria, RQL provides support for path expressions (schema and instance navigation), union, difference, quantification, aggregation, namespace querying, lexical space querying, value space querying, entailment and partial support for optional path expressions, reification, collections and containers. For the generation of RQL queries we can use GRQL [69], an application-independent graphical user interface (GUI). Finally, we choose RQL since it is considered to be the most complete RDF query language in comparison to other popular ones (RDQL, Triple, SeRQL, Versa, N3), according to elicitations extracted from recent evaluations ([70], [71]) and has the additional advantage of disallowing cycles in a given subsumption hierarchy. The last is crucial, as long as we have isA links between different peers, and the cycles upon the schema are possible.

The clustering of peers according to semantic information would help us to our proposed routing strategy in the process of discovering the relevant peers to the specified query. *Semantic Overlay Networks (SONs)* appear to be an intuitive way to cluster together peers sharing the same schema about a community domain or application model. We assume that each Super-Peer is joined to at least one specified SON and is responsible for it. We can use a clustering policy for the joining of peers to the specific SON similar to the approach based on *Semantic Overlay Clusters* (SOCs see subsection 2.3.3). We suppose that each Super-Peer deal with some topics, which characterizes its Semantic Overlay Networks.

When a new peer requests to join in the SON of a specific Super-Peer, the last applies the defined *clustering policy* and accept it in its cluster or deny it. By this way semantically irrelevant peers could not be joined in the same SON, and as result they could not be in the specific cluster that formulates the specific Super-Peer. One clustering policy that can be used in our context is the matching of the candidate peer's terms of its taxonomy with the topics that the Super-Peer is dealt with. The initial role of Super-Peers is to collect the RDF/S schemas of the peers that are responsible for its cluster. This will help to the routing phase, as we should see in order to decide the relevant peers where the query has to be routed. For mediation purposes, we can consider that each RDF/S schema is defined as a view (Local-as-View) on some global schema that each Super-Peer holds.

3.2.2 Query Routing Algorithm

Let's assume a number of Super-Peers that have in their responsibility a cluster of peers according to its corresponding Semantic Overlay Networks. For simplicity let's suppose that in each Super-Peer's responsibility is only one SON (see an example in Figure 13 below). This assumption doesn't affect the routing steps of our suggested strategy, which in general can be applied for more than one SON for each Super-Peer.

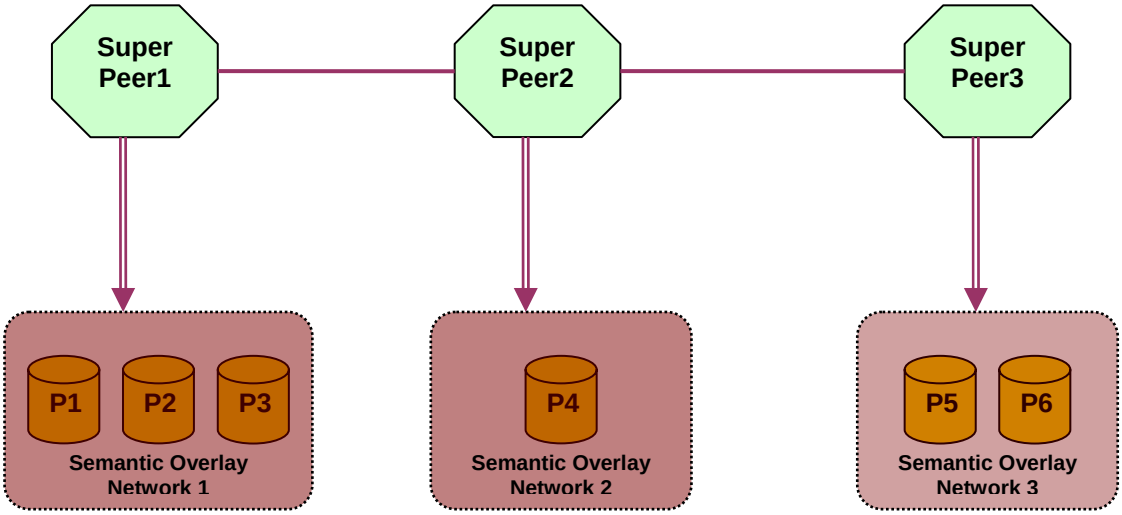

Figure 14: An example of a P2P network built upon our proposed architecture and with regard to our suggested routing context

Let' assume that peer P1 (which belongs to Semantic Overlay Network 1 and its responsible is Super-Peer1) makes a query Q. The steps of our routing algorithm are the following:

- We find at first the responsible Super-Peer for P1 which is in this example Super-Peer1.
- This responsible Super-Peer examines all the RDF/S schemas of the peers that belong to its cluster and finds out all the relevant peers according to the matching of their terms (at schema level) and their properties (at instance level) with the query predicates.
- If the responsible Super-Peer found relations between peers at different clusters (i.e. through the use of in relation isA links) then add to the set of relevant peers, these ones that also match to the query predicates according to their schema and instance level information.
- Then the final set of relevant peers and its corresponding Super-Peers are returned in order to be processed by our suggested processing algorithm.

We should point out that the definition of relevant peers can be flexible according to the needs of the applied peer-to-peer network. Surely a relevant peer must match at Schema-Level with at least one property and one relation. The strict definition corresponds to a case where all properties and relations (at schema-level)

are defined on the peer's RDF/S schema and the requested conditions upon their values are valid.

3.2.3 Advantages of Query Routing Strategy

In general if we have an RDF/S Schema to describe each peer's knowledge base then our routing technique can be applied to heterogeneous databases. The only thing, we should need to support in order to get back the results, is a kind of wrapper that would undertake the transformation of the query to a suitable query language for each database in order to proceed the matching process. In addition, for Semantic Web this technique is surely applicable since everything can be thought as a RDF description and there is interoperability between the common standards (XML, RDF/S, OWL).

We can obviously conclude that we abuse the network topology, that's why the Super-Peers have an important role to all the suggested routing process. Thus, a meaningful advantage of this query routing context is that peers are grouped based on the semantics of their stored data. Thus, since queries are routed according to the same semantically-based classification policy, required results to each query are found faster and only from relevant to the query peers which are considered. We should mark that peers that have few or no results considering a given query will not be contacted, since their classification will assign them to different Semantic Overlay Networks, thus avoiding wasting processing and communication resources on that requests.

Furthermore, the flexible use of taxonomies, their corresponding RDF/S schemas and the links among them facilitate the routing process which does not have to take care of complicated and probably time-consuming tasks such as index maintenance and updates. The two-way links obviously helps in the case when we are looking for a term that does not exist at a peer, but this peer "knows" from a two-way link that one of its terms is related with the asked term but is contained to another peer.

Finally the harmonic combination of the building blocks of our query routing context provides a routing strategy that supports self organization and distribution of peers, simple, accurate and not complicated procedural steps. Therefore each query is routed only to a set of relevant peers with low bandwidth consumption across the

applied p2p network where the query will be processed by the proposed processing strategy which is presented in the next section.

3.3 Query Processing Strategy

3.3.1 General Features, Issues and basic steps

Our query routing technique as is presented in the previous section returns a set of relevant peers to the query. At this set we want to apply our query processing technique. The new trend in query processing is the adaptation of top-k retrieval algorithms in order to get back the results quickly and without any large processing cost. A top-k retrieval algorithm generally returns the best k results (top-k results) according to a given criterion. The top-k results and the applied criterion depend upon the applied scoring technique upon the data of peers. In the section 2.4 (Query Processing in P2P Networks) we concluded that two are the dominant approaches for large-scale distributed networks like P2P networks: the Marian et. al and the Three Phase Threshold Approach. We have to compare their characteristics and finally decide which one is the best and we should adapt it under our defined peer-to-peer environment. Having in mind that the scalability of a peer-to-peer network depends upon the communication cost of each peer, we should denote a cost analysis of our suggested top-k processing algorithm. Also we have to define our scoring technique and the use cases of top-k processing that arise from different scoring techniques.

Before we start to analyze all these general features and issues that comprise our suggested processing strategy we will define the basic abstract steps for each query that is going to be processed as they depicted in Figure 14 below. First the top-k query is made by one participant peer of the whole peer-to-peer network. The query routing algorithm takes on to find the relevant peers to the query and sends the last to them. At the set of relevant peers to the query the scoring technique is applied upon its data, so all the candidate top-k objects are returned with its specific value. In particular (Object, Score) pairs are returned as input to the top-k processing algorithm. Finally the last returns the top-k objects and their required data from the peers where they are located to the peer which originates the query.

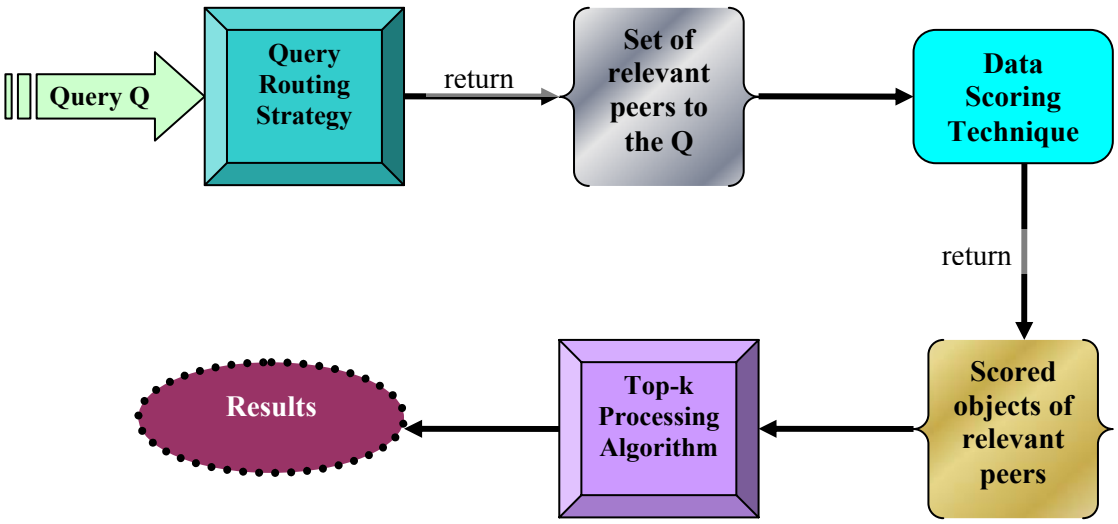

Figure 15: Basic abstract steps to processing of a query

3.3.2 Selection of Top-k basic algorithm

In this subsection we examine in detail the (p)Upper algorithm [63, 64] that come from Marian et. al. approach and the Hybrid Threshold (HT) algorithm [15] that come from the Three Phase Threshold Approach. As we have already mentioned these two approaches are the only ones that support of estimating data distributions without a-priori knowledge. This characteristic has great meaning for score-based algorithms because:

- By estimating data distribution, these algorithms examine even in extreme cases of scoring and have the ability to prune non-eligible objects.
- The advantage of estimation without a-priori knowledge conduces in a fully dynamic strategy, which works well even if the number of peers is small and the knowledge about them is limited.

We should remind here that approaches like Nejdl et. al [53, 62] in top-k processing cannot work efficiently since they do not support this characteristic and requires enough information to routing indices, whereas in the Probabilistic – Histograms approach [57, 58, 59, 60] the authors assume that each participating peer already owns histograms for all of its neighbours and the queried attributes.

Therefore, the common advantage of HT and (p)Upper algorithm is the estimation of data distribution without a-priori knowledge. However, HT has the additional advantage of standard number of rounds (three + one as we will see at the presentation of HT in the next subsection) which can be executed in a distributed way by peers, except from one case where a universal threshold is required by all participating peers. On the contrary, (p)Upper only one source can be accessed at a time, which means that at each round each peer must wait for the others to send its score. To reduce this limitation pUpper enables parallel top-k processing and emphasize only on reducing query response time through the use of queues in order to gain the lost time from the delay of accesses at each peer (source). But, in a widely distributed scenario (p)Upper may incur in a potentially unbounded number communication (messages) rounds. In addition if the parameter L which indicates the length of the random-access queues is not chosen correctly pUpper might perform "useless" probes. Thus, it is not easy and flexible enough to use the queues in pUpper algorithm.

The HT needs sorted access lists of scores, while (p)Upper needs at least 1 sorted source. Furthermore, (p)Upper seems to send more but smaller messages than HT which sends a standard number of bigger messages per node (including partial scores etc.) Finally we choose the Hybrid Threshold because it fits well in a 2-tier distributed system. In our peer-to-peer topology we have assumed that there are peers and Super-Peers, so we can think of it as 2-tier architecture. Surely some changes are needed to do for adapting HT in our peer-to-peer scenario and we will talk about them after we present the original version of Hybrid Threshold algorithm as presented in [15].

3.3.3 The Hybrid Threshold Algorithm (HT)

3.3.3.1 The original applied context

The authors of [15] deal with the problem of answering top-k queries efficiently in distributed networks and they presented the original version of Hybrid Threshold. They consider *Content Distribution Networks (CDNs)*, which are deployed by many companies to avoid network congestion. CDNs typically consist of cache servers scattered around the globe for caching bandwidth-intensive objects from the

original server such as images and video clips. This enables fast web and streaming media applications. When a request is sent to the original server, it is redirected to one of the cache servers which is closer to the client and/or can serve data faster. Effective monitoring of activities (by a central manager) over CDNs ensures successful content distribution. One such monitoring task is a top-k query, e.g., *"what are the top-k most popular URLs across the entire CDN?"*?

A naive approach to answer such a query is to have each cache server send the access statistics about all objects to the central manager. However, this incurs significant bandwidth consumption if the number of objects at each cache server is large. Exactly for this reason the authors of [15] suggested a family of bandwidth efficient algorithms for processing such top-k queries in a distributed environment with Hybrid Threshold to be the optimal under all testing cases. Thus the authors formalize the problem of top-k query processing in distributed systems by abstracting the above CDN example.

They assume that there are m nodes and one single central manager in the specified distributed system (CDN). Each node i is connected to the central manager and maintains a list of pairs (O, $S_i(O)$) where O is an object and $S_i(O)$ is the score of the object. Also they assume that objects in each list are sorted in the descending order of their scores. If an object does not appear in the list of a node, its score in that list is zero by default. The central manager initiates a top-k query and finally retrieves objects from the network with the k highest f(S1(O),...,Sm(O)) where f is a monotonic function (such as the summation function SUM) to compute the overall score of an object. For simplicity, the authors assume the sum function but in practice it could be a weighted sum to account for the relative importance of cache servers.

3.3.3.2 The original HT

For the above distributed (original) context the required steps of HT are the following:

Phase 1:

- Each node sends its top-k objects to the central manager. The central manager then calculates the partial sums for all objects seen so far and identifies the objects with the k highest partial sums. For an object O, the partial sum $S_{psum}(O) = S'_1(O) + \ldots + S'_m(O)$ where $S'_i(O) = S_i(O)$ if

O has been reported by node i to the central manager, and $S'_i(O) = 0$ otherwise.

Phase 2:

- The central manager broadcasts the list L to all the nodes in the network and T = τ1/m as well.
  - L = list of the top-k object IDs from the partial sum list.
  - τ1 = phase1 bottom: the kth highest partial sum.
  - m = the number of nodes.
- Upon receiving the list L, for each object Oj in L: node i finds its local score Vij and determines the lowest local score Slowest(i) among all the k objects in L. (if Oj does not occur in the local list, Vij = 0)
- Then node i sends the list of local objects whose values are >= Ti = max(Slowest(i),T) to the central manager.
- Now the central manager calculates the partial sums for all the objects seen so far, and identifies the objects with the k highest partial sums.
  - Let us call the kth highest partial sum "phase-2 bottom" and denote it by τ2.

Phase 3: (patch phase if necessary)

- The central manager checks if the threshold from node i, Ti in phase 2 is greater than Tpatch = τ2 /m.
- If so, the central manager will send Tpatch to node i as the threshold and ask it to send all the objects whose scores are no less than Tpatch
- Now the central manager calculates the partial sums for all the objects seen so far, and identifies the objects with the k highest partial sums
  - Let us call the kth highest partial sum "phase-3 bottom" and denote it by τ3.

- Then the central manager tries to prune away more objects: It calculates the upper bounds of the objects seen so far using: Usum(O)= S'1(O) + S'2(O), +… S'm(O), where

- S'i(O) = Si(O) if O has been reported by node i
- S'i(O) = min(Ti, Tpatch) otherwise.

- Then the central manager removes any object Oj from the candidate set whose upper bound is less than τ3.

Phase 4:

- The central manager sends the top-k object candidate set to each node and each node in turn sends the scores of these objects to the central manager.
- Hence, the central manager can calculate the real scores for these objects and then identify the exact top-k objects.

At this point we have to mention that although the HT has four phases, we classify it under the approach of Three Phase Threshold algorithm. This occurs because the firstly three are the basic phases and secondly it does not change anything at the basic characteristics of the algorithm, comparing with the other algorithms of the same family. The patch phase is needed for each node where Tpatch < Ti or Tpatch = Ti. But if Tpatch > Ti for every i, there is no need for this patch phase because all top-k object candidates have been considered according to their lower bounds that have been calculated at Phase 2 of the algorithm. Finally the authors can prove that HT algorithm correctly returns the exact top-k objects for any data distribution in each node of a two-tier distributed system.

3.3.3.3 Evaluation of HT

The authors that suggested HT in [15] made some experiments to prove the good performance of their algorithm comparing it with other algorithms that belong to the same family (Three Phase Threshold Approach). The performance metric they used was the bandwidth consumption. They were mainly concerned with the number of (Object, Score) pairs sent from nodes to the central manager since it is the dominant factor in bandwidth consumption. They assume as well that the computation cost in each node was negligible while the communication cost among nodes dominates the query response time.

Some experiments were upon synthetic data sets that used Zipf distribution [72] and a Zipf factor. They use a random model for the scores of objects. As the next Figure 15 shows a representable result of this specific experiment where the nodes m are 100 and the Zipf factor = 0,5. At the horizontal axis of the chart the used values of k for a specific sample query were: k = 5, k = 10, k = 26, k = 50, k = 100 while at the vertical axis the number of (Object, Score) pairs is shown. At this specific experiment the algorithms TPUT, TPOR and HT were tested.

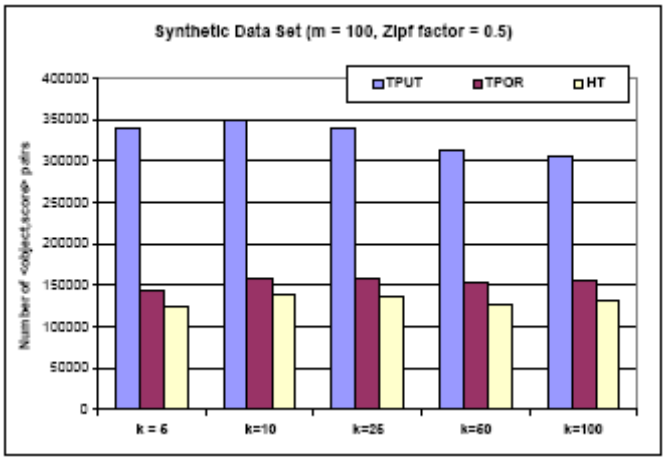

Figure 16: Performance comparisons over a synthetic data set

From this figure we can conclude that HT sends the fewer (Object, Score) pairs and has the better performance against the others (TPUT, TPOR). Also another general and meaningful conclusion is that all these algorithms that belong to the specific family return approximately the same number of (Object, Score) pairs for different values of k. Surely in a wide distributed environment we should use and other performance metrics to prove the good performance of HT. Therefore we could measure how the execution time of the algorithm is affected as the number of nodes increased. In addition we should remind that in the original applied context of HT exists only one single central manager that collects the results and m nodes that contribute to them.

3.3.4 The HT-p2p: A Hybrid Threshold algorithm for a Super-Peer – Based P2P

3.3.4.1 The HT-p2p context and basic features

In this subsection (3.3.4) we present an extended version of the Hybrid Threshold adapted to our peer-to-peer scenario called *HT-p2p.* This new algorithm can be applied in any Super-Peer based peer-to-peer network as an efficient top-k processing algorithm. HT assumed that there are m nodes and one single central manager in a Content Distribution Network (CDN). In HT-p2p we assume that there is a large number of Super-Peers (s Super Peers) which are responsible for a number of peers (m peers). In particular following our suggested routing context each Super-Peer has a cluster of peers (let's assume that we have m peers at each cluster). Our applied network is not a CDN, but a peer-to-peer network. Each Super-Peer is not just a connector where its corresponding peers are connected and return its required data at each phase of the algorithm. In HT-p2p each participant Super-Peer saves some intermediate results which help it to prune some steps of the basic algorithm.

Before describing in detail all the required steps for HT-p2p we should denote the roles that peers are taking on the execution of the algorithm. The peer that makes the original query across our peer-to-peer network called originator peer. Respectively its responsible Super-Peer plays the role of Originator Super-Peer. Each peer can be a contributor peer or a non-contributor. Finally a collector Super-Peer collects the required data (i.e. Object, Score pairs) and finally returns them to the originator Super-Peer. These roles are analyzed below:

- **Contributor peer**: a peer that participates to the execution of HT-p2p and contributes to the top-k results. This should be a relevant peer to the query as returned from the query routing strategy. Contributor peer sends the required (Object, Score) pairs to the specified Super-Peer.
- **Non Contributor peer:** this kind of peer at the specific running instance of HT-p2p does not participate and it is in practice inactive.
- **Originator peer:** it is the peer that makes the original query across our peer-to-peer network.
- **Originator Super-Peer:** the corresponding responsible Super-Peer for the originator peer. Sends the final top-k results to the last.

- **Collector Super-Peer:** it is the Super-Peer that executes the specific running instance of HT-p2p. It collects all the intermediate results from all the contributor peers and finally returns them to the Originator Super-Peer.

Each time a query is processed there is only one originator peer and its corresponding originator Super-Peer. The collector Super-Peer starts and runs the specific instance of HT-p2p. It could be the originator Super-Peer, or anyone else. For the selection of the collector Super-Peer we could take into account the number of contributor peers or the number of the relevant objects. We can choose the originator Super-Peer to be the collector as well in order to have one less message at the end of the algorithm where the collector sends to the originator the final top-k results. At each specified running instance of HT-p2p all the contributor peers participate which are determined by the query routing strategy by detecting the relevant peers to the top-k query. In this way we know which ones are the non-contributor peers.

For HT-p2p we use the same query model, as all authors of Three Phase Threshold Approach [15, 65]. Therefore we assume that each peer maintains a list of pairs (O, $S_i(O)$) where O is an object and $S_i(O)$ is the score of the object. The objects in each list are sorted in the descending order of their scores. If an object does not appear in the list of a peer, its score in that list is zero by default. Each specified Super-Peer initiates a top-k query and finally retrieves objects from the network with the k highest $f(S_1(O),...,S_m(O))$ where f is a monotonic function, to compute the overall score of an object. Let's assume that the monotonic function is the SUM function. We should remind at this point that each object is scored accorded to the selected scoring technique which determines the applied monotonic function. Each object can be thought as a RDF resource if we are talking about RDF/S data instances. For relational databases each object can be thought as a tuple.

3.3.4.2 The HT-p2p Algorithm

As the routing strategy has returned the set of ranked objects of relevant (contributor) peers across the peer-to-peer network, an instance of HT-p2p is ready to run starting by the collector Super-Peer. The processing steps of HT-p2p are the following:

Phase 1:

- Each contributor peer **sends** its top-k objects to the collector Super-Peer. The last then calculates the partial sums for all objects seen so far and identifies the objects with the k highest partial sums. The collector Super-Peer stores all the intermediate results of this phase (seen objects, their scores, and their partial sums).
- *For an object O, the partial sum* $S_{psum}(O) = S'_1(O) + \ldots + S'_m(O)$ *where* $S'_i(O) = S_i(O)$ *if O has been reported by peer i to the Super-Peer, and* $S'_i(O) = 0$ *otherwise. An object has been reported by a peer if it has been sent with its score to a Super-Peer at least one time, so it has been stored.*

Phase 2:

- The collector Super-Peer **broadcasts** the list L and the threshold T = τ1/m as well to all the contributor peers in the p2p network.
  - L = list of the top-k object IDs from the partial sum list.
  - τ1 = "phase1 bottom": the kth highest partial sums.
  - m = the number of peers at the specified cluster of Super-Peer.
- Upon receiving the list L, for each object Oj in L: peer i finds its local score Vij and determines the lowest local score Slowest(i) among all the k objects in L. If Oj does not occur in the local list then Vij = 0
- Then peer i **sends** the list of local objects whose values are >= Ti = max(Slowest(i),T) to the collector Super-Peer.
- Now the Super-Peer calculates the partial sums for all the objects seen so far, and identifies the objects with the k highest partial sums.
  - Let us call the kth highest partial sum "phase-2 bottom" and denote it by τ2.
- The collector Super-Peer denoted Tpatch = τ2 /m. If (Ti < Tpatch) the collector Super-Peer **sends** these objects (with the k-highest partial sums) as top-k objects to the

originator Super-Peer which ***return***s them to originator peer which made the original top-k query.

- But if (Ti > Tpatch) where Tpatch = τ2 /m then two additional phases (Phase 3, Phase 4) are needed for each peer i where the above condition is true.
- The collector Super-Peer stores all the intermediate results of this phase (seen objects, their scores, and their partial sums).

Phase 3: (patch phase if necessary)

- The collector Super-Peer **sends** Tpatch to peer i as the threshold and ask it to **send** all the objects whose scores are no less than Tpatch.
- Now the Super-Peer calculates the partial sums for all the objects seen so far, and identifies the objects with the k highest partial sums
    - Let us call the kth highest partial sum "phase-3 bottom" and denote it by τ3.

- Then the Super-Peer tries to prune away more objects by calculating the upper bounds of the objects seen so far and have been stored till now.
- *An upper bound for an object O (Usum(O))is calculated by the formula: Usum(O)= S'1(O) + S'2(O), + … S'm(O), where*
    - *S'i(O) = Si(O) if O has been reported by node i*
    - *S'i(O) = min(Ti, Tpatch) otherwise.*
- Then the Super-Peer removes any object Oj from the candidate set whose upper bound is less than τ3 and return the top-k candidate set.

Phase 4 (necessary if we run Phase 3)**:**

- Since the collector Super-Peer stores the intermediate results (seen objects, their corresponding scores and partial sums of them) at this phase it just calculate the real scores for the top-k candidate set as it returned from the previous phase and then identify the exact top-k objects.

- Finally it **sends** the top-k objects to the originator Super-Peer which ***returns*** them to originator peer which made the original top-k query.

As we could note HT-p2p is an extended and improved version of HT adapted under our peer-to-peer scenario. It assumes a Super-Peer based architecture where the collector Super-Peer runs an instance of the algorithm upon the contributor peers which can belong to different clusters and they are retuned as relevant from the query routing strategy. The advantage of HT-p2p is that it can return in some cases (when we don't have any patch phase) the final results in phase 2 because of the storing capability of the intermediate results. For the same reason in phase 4 of HT-p2p compared with phase 4 of HT does not need to request from peers to send their scores since they have been saved at the previous phase. As we can see from each phase of HT-p2p at each contributor peer they can be executed in parallel except from the case where the calculation of phase-2 bottom is needed for the denotation of Tpatch. At this specific point peers should wait until the phase 2 bottom is defined.

Having in mind that each Super-Peer in real conditions can have under its cluster many thousands of peers it is desirable sometimes for performance and scalability reasons to host one running instance of HT-p2p at each relevant Super-Peer which should be executed independently of each other and in a distributed way. Then we should combine the results from all Super-Peers and calculate the real scores for their "top-k" objects in order to find the k-highest ones which denote the final top-k results. The last process requires from Super-Peers to receive scores from their corresponding peers that maybe were not sent at the real execution of the specific HT-p2p's running instance. This approach introduce a modified version of HT-p2p we call it *HT-p2p+*.

We can observe that by applying HT-p2p+ we have a more distributed processing policy which promises better performance when we have a large number of relevant peers at different Super-Peers. On the other hand for each running instance we have an extra processing cost and probably communication cost (calculation of real scores of unseen objects) in order to finally combine the results at the collector Super-Peer. Obviously according to the selected version of the algorithm there is a trade-off between performance and network consumption which is a common fact at

distributed environments like peer-to-peer networks. In the next subsection we present in which points HT-p2p+ differs from HT-p2p.

3.3.4.3 The HT-p2p+ Algorithm

In HT-p2p+ for Super-Peer an additional role has to be defined. The role of *contributor Super-Peer,* which contributes to the final top-k results by applying a running instance of HT-p2p across its relevant peers. Similarly to the definition of non contributor peers, the *non contributors Super-Peers* don't run any instance of HT-p2p since they don't have any relevant peers according to the routing algorithm. Specifically as the routing strategy has returned the set of ranked objects of relevant (contributor) peers for each corresponding (contributor) Super-Peer an instance of HT-p2p+ is ready to run. If the originator Super-Peer has the role as well of contributor Super-Peer, then we select to give the additional role of collector. Otherwise, a collector Super-Peer can be anyone of the contributor Super-Peers. The role of collector Super-Peer in HT-p2p+ is to collect all the top-k results from all the running instances of HT-p2p+, combines them and finally returns them to the Originator Super-Peer.

Except from these roles HT-p2p+ differs from HT-p2p in some points of executed steps. At all phases the active role of Super-Peer is the contributor Super-Peer and not the collector Super-Peer. Thus the contributor Super-Peer interacts with the contributor peers at each phase and stores all the intermediate results of this phase (seen objects, their scores, and their partial sums). The collector Super-Peer acts at the end of phase 2 if we don't have any patch phase and at the end of phase 4 of HT-p2p+. At these two cases, the specific contributor Super-Peer sends the results to collector Super-Peer. Then, the collector Super-Peer combines all the results from the contributor Super-Peers and finally returns the top-k objects to the originator Super-Peer.

We should analyze further how the combination of results is made, because it could denote a different extra phase. Assuming that each contributor Super-Peer has sent its top-k object set to the collector Super-Peer. The last has to choose the objects with the k-highest real aggregate scores among all top-k object sets. Thereby, it accumulates all the discrete candidate objects and sums their real scores. It is possible, the collector Super-Peer to ask from each contributor Super-Peer to calculate the real

score for each object if it does not contained in its top-k object set. In this case is needed all contributor peers to send their scores to its contributor Super-Peer (as long as they have not sent them at the real execution of HT-p2p+) in order to calculate the real score for each object. The objects with the k-highest real scores are the final top-k objects and they are sent to the originator Super-Peer.

3.3.4.4 Data Scoring and Use Cases of HT-p2p

The scoring technique in HT-p2p is applied only to the set of relevant peers, in order to reduce the pre-processing cost of the whole processing strategy (see Figure 14 above). According to the scoring technique and based on our suggested routing strategy we can define and specify more than one use case of HT-p2p. We have assumed that each query across the peer-to-peer network is a top-k query, so we look for top-k objects which have the k highest overall scores. HT-p2p takes as input sorted lists of (Object, Score) pairs. There is one list for each contributor peer. The meaning of each score at the scoring technique represents the specified use case.

If we consider a peer-to-peer network where each peer is autonomic to rank its objects, then the same object would probably has different scores at different peers. We can think an example of a scenario where peers are ranking movies according to their preferences and we are looking for the movies with k highest overall score. In this case a scoring function could be a monotonic function (such us SUM).

Another scenario could be the case, where we have a top-k query upon some attributes, and the contributor peers don't have information about all attributes. Let's assume that we are looking for the top-k hotels that have: price < 100 Euros and rating>3 stars and distance<5 km from general hospital. In this case price, rating and distance are the specific attributes of the top-k query. Then some peers maybe do not have in their database information about an attribute, for example distance from general hospital. Our scoring technique could put zero in this attribute and could sum the others attributes to compute the final score at each peer. Furthermore we could use a weighted monotonic function for the computation of the final score of each object: *Final Score = w1*s1 + w2*s2 + w3*s3* where w1,w2,w3 are the predefined weights for each attribute according to the preferences of originator peer. Since the attributes are numerical we could denote that their corresponding scores s1, s2, s3 are ranked

according to how close are their values to these that are requested at the original top-k query.

The last and perhaps the trivial and simple one is the case where each peer ranks the same objects with the same score by applying the same scoring function. Then a top-k query has meaning of use if peers don't contain exactly the same objects. Because in the case where all peers have the same objects with the same scores, we don't need to make a top-k query, but retrieve the highest k scores just from one peer. At the specified simple case, we could use any kind of monotonic function to rank objects at each peer. HT-p2p will return in this case as well the top-k objects with the highest values of the monotonic function across the peer-to-peer network.

3.3.4.5 Cost Analysis of HT-p2p / HT-p2p+

As we can observe in each phase some (Object, Score) pairs are transmitted from the contributor peer to the corresponding Super-Peer. This number varies according to the data distribution and could be a performance metric for the bandwidth consumption as the authors of [15] suggested and as they mentioned at the evaluation of HT (subsection (3.3.3.3). Before testing HT-p2p under real conditions (see next chapter: Implementation) it is useful to examine in detail the number of standard messages that are required at each phase of it since it is designed and applied for peer-to-peer networks make us. This try is termed *network cost analysis*. For large-scale distributed networks we are not interested on how many phases are required, but how many messages are needed to send from peers to Super-Peers and vice versa.

At phase 1 each contributor peer sends one message to its corresponding Super-Peer. At phase 2 each contributor Super-Peer broadcasts another message to all of the peers in its cluster. Also each peer after receiving the last message from the specific Super-Peer sends one more to it. If we don't have any patch phase we need one additional message per contributor Super-Peer which it is sent to the collector Super-Peer. The last message contains the final top-k results and is sent from the collector Super-Peer to originator Super-Peer. In fact if the collector Super-Peer is contributor as well, we need one less message. The originator Super-Peer in its turn returns the top-k final object set in a message to its originator peer.

If we have a patch phase then each contributor Super-Peer is needed at first to send one message to each contributor peer that must execute the patch phase. The last after receiving this message has to send one more message to its contributor Super-peer. Then we need one additional message per contributor Super-Peer which it is sent to the collector Super-Peer. At phase 4 the last message contains the final top-k results and is sent from the collector Super-Peer to originator Super-Peer. The last in its turn returns the top-k final object set in a message to its originator peer.

Assuming that we have w contributor Super-Peers (case of HT-p2p+), m contributor peers at each cluster, n peers that have to run patch phase (n<m) and the collector Super-Peer is contributor as well we show the required messages at each phase at the Table 2 below. We should mark that, if we run just one instance across the peer-to-peer network (case of HT-p2p) then we don't have any contributor Super-Peers but only one collector peer that does all the job. In fact in this case our contributor Super-Peer is the collector, so we consider in this case that w = 1. The final number of required messages for this completed scenario to transfer across the peer-to-peer network is given by the following type:

$$\textit{Total Number Of Messages in HT-p2p+} = 3m * w + 2n * w + 2.$$

$$\textit{Total Number Of Messages in HT-p2p} = 3m + 2n + 2.$$

Phase	Sender	Number of messages
Phase 1	contributor peer(s)	m * w
Phase 2	contributor Super-Peer(s)	w * m
Phase 2	contributor peer(s)	m * w
Phase 3	contributor Super-Peer(s)	w *n
Phase 3	contributor peer(s)	n* w
Phase 4	collector Super-Peer	1
Phase 4	originator Super-Peer	1

Table 2: Required messages for a completed executing scenario of HT-p2p+

3.3.4.6 An example of HT-p2p

We assume that we have the collector Super-Peer: SP1. Let's assume that the relevant peers according to the routing strategy are: Peer1, Peer2, and Peer3. Thus these peers are the contributors to a sample query at this example. Let's assume that the originator peer is peer3, so SP1 is also the originator Super-Peer and we are looking for a top-2 query (k = 2). Table 3 below is shown the (Object, Score) pairs at each peer of SP1. Peer1 has 10 relevant to the query and top-k candidate objects that have been resulted from the routing strategy and they are ranked and sorted in the descending order at column1. Peer2 has 9 top-k candidate objects and Peer3 has 6 top-k candidate objects. The execution of HT-p2p has to return the top-2 objects with the highest overall scores without examining all objects of each sorted lists at each peer.

Peer1	Peer2	Peer3
(O5, 21)	(O4, 34)	(O3, 30)
(O2, 17)	(O1, 29)	(O4, 14)
(O4, 11)	(O_O, 29)	(O_O, 9)
(O3, 11)	(O3, 26)	(O5, 7)
(O6, 10)	(O9, 20)	(O2, 1)
(O7, 10)	(O5, 9)	(O8, 1)
(O11, 8)	(O14, 5)	
(O12, 6)	(O16, 2)	
(O15, 6)	(O13, 1)	
(O13, 4)		

Table 3: (Object, Score) pairs at each peer of SP1

The executing steps of HT-p2p for the sample top-2 query of the example are the following:

Phase 1: *Peer1 sends its top-2 objects with its corresponding scores to SP1: (O5, 21), (O2, 17). Peer2 sends respectively (O4, 34), (O1, 29) to SP1 and Peer3 sends (O3, 30), (O4, 14). Then SP1 calculates the partial sums (S_{psum}) for all seen objects: $S_{psum}(O5) = 21$, $S_{psum}(O2) = 17$, $S_{psum}(O4) = 48$, $S_{psum}(O1) = 29$, $S_{psum}(O3) = 30$. The k=2 highest partial sums are belong to O4, O3 and their value is 48, 30 correspondingly. SP1 stores all the intermediate results of this phase (seen objects, their scores, and their partial sums).*

Phase 2: *According to the results of Phase 1 τ1 = 30 (since it is the kth highest partial sum) and the list L contains O4, O3. Thus Super-Peer SP1 broadcast the list L = {04, 03} and the threshold T =10 since is equaled with the fraction τ1 / m where m = 3 (number of contributor peers for this Super-Peer). Then each peer firstly determines its lowest score upon the objects of the list L (04, 03). At peer1 the lowest local score is 11 and it is come both from object O3 and O4. At peer2 the lowest local score is 26 and come from object O3, where at peer3 the lowest local score is 14 and come from object O4. Secondly, each peer calculates its threshold Ti = max(Slowest(i),T) and sends its objects whose values are greater than Ti to SP1. For peer1 T1=11 so it sends objects (O5, O2, O4, O3) to SP1 with their scores, for peer2 T2=26, so it sends objects (O4, O1, O_0, O3) with their scores and for peer3 T3=14 so to SP1 are sent only the pairs (O3, 30), (O4, 14). Afterwards Super-Peer SP1 calculates the partial sums for all objects seen so far and identifies the objects with the k highest partial sums. The partial sums are:* $S_{psum}(O5) = 21$, $S_{psum}(O2) = 17$, $S_{psum}(O4) = 59$, $S_{psum}(O3) = 67$, $S_{psum}(O1) = 29$, $S_{psum}(O_O) = 38$. *SP1 stores all the intermediate results of this phase (seen objects, their scores, and their partial sums). Thus, the 2 highest are* $S_{psum}(O3)$ *and* $S_{psum}(O5)$ *where the last is equal with τ2 since it is the kth. Therefore Tpatch = τ2 / m = 59 / 3 = 19,6 ⇒ Tpatch = 19. Now SP1 checks if there is a need for patch phase at each peer by checking the condition Ti > Tpatch. For peer1 and peer3 there is no need of patch phase because their thresholds (T1=11, T3=14) are not greater than Tpatch. But for peer 2 we need to execute a patch phase since T2=26 > 19. P1 stores all the intermediate results of this phase (seen objects, their scores, and their partial sums) at goes to the next phase.*

Phase 3: *SP1 requests from peer2 to send all its objects that are no less than Tpatch = 19. Thus peer2 sends {O5, O14, O16, O13}. Now the current seen objects at SP1 are:* $O_{o,,}$ *O1, O2, O3, O4, O5, O13, O14, O16. Their corresponding partial sums are:* $S_{psum}(O_O) = 38$, $S_{psum}(O1) = 29$, $S_{psum}(O2) = 17$, $S_{psum}(O3) = 67$, $S_{psum}(O4) = 59$, $S_{psum}(O5) = 21$, $S_{psum}(O13) = 1$, $S_{psum}(O14) = 5$, $S_{psum}(O16) = 2$. *Thus τ3 = 59 since the kth highest partial sum is come again from O4. For these objects SP1 calculate their upper bounds at peer2:* $S'(O_o) = 29$, $S'(O1) = 29$, $S'(O2) = 19$, $S'(O3) = 26$, $S'(O4) = 34$, $S'(O5) = 9$, $S'(13) = 1$, $S'(14) = 5$, $S'(16) = 2$. *Then SP1 prunes* O_o, *O1,*

O2, O5, O13, O14, O16 objects from the top-k candidate set since their upper bounds are less than τ3.

Phase 4: *Since the top-k candidate set from phase 3 contains exactly k=2 objects there is no need to calculate the real scores for these objects to determine the highest ones, so SPI which is the originator Super-Peer return O3 and O4 to peer3 (originator peer) as top-k objects.*

3.3.5 Advantages of Query Processing Strategy

Our query processing strategy takes as input (Object, Score) pairs which are sorted in the descending order. These pairs are returned by the scoring technique which takes as input the objects from relevant (contributor) peers that are returned by the query routing algorithm. From this control flow as it is shown in Figure 14 above we can conclude that our query processing strategy collaborates with our query routing strategy, but it is independent of it. In other words any query routing technique can be applied upon our proposed Super-Peer RDF/S peer-to-peer network in order to define the top-k candidate objects that come from the specific relevant peers and determine the role of each peer according to the considerations of HT-p2p.

In general our suggested query processing strategy can be applied to any Super-Peer based peer-to-peer network. HT-p2p works well for any number of peers and Super-Peers that are organized in clusters. Thus we talk about a scalable processing approach which is crucial for large scale distributed networks. For us, following the proposed query routing context these clusters formulate semantic overlay clusters, where the semantically relevant peers are grouped together.

Furthermore, each peer has a discrete role in the processing strategy and at each phase each contributor peer runs independently from each other and in a distributed manner. For the same reasons and in order to reduce the delay cost at phase 2 of HT-p2p, where all the contributor peers are waiting for a universal threshold we have suggested the role of contributor Super-Peer which runs an instance of the modified HT-p2p algorithm called HT-p2p+ upon its clustered contributor peers and returns its results to the collector Super-Peer. The last undertakes to combine all intermediate top-k results and to return the final ones. Therefore we could

say in general that HT-p2p and HT-p2p+ act as a distributed algorithm which takes advantage the role of all participant peers.

As we have mentioned the scoring technique in HT-p2p is applied only to the set of relevant peers, in order to reduce the pre-processing cost of the whole processing strategy (see Figure 14 above). According to the scoring technique and based on our suggested routing strategy we have defined three use cases of HT-p2p. This means that our processing technique can be applicated into many scenarios that takes place in peer-to-peer networks.

Another advantage of HT-p2p is that it can return in some cases (when we don't have any patch phase) the final results at phase 2 because of the storing capability of the intermediate results. By this way HT-p2p prunes two phases under specific conditions. The rest advantages come from the characteristics that HT-p2p "inherit" from its ancestor HT. Thus, it has standard number of rounds and estimates data distribution without a-priori knowledge. Hence, it can examine even extreme cases of scoring with too relevant or too irrelevant data and has the ability to prune non-eligible objects. Finally the estimation without a-priori knowledge conduces in a fully dynamic strategy, which works well even if the number of peers is small and the knowledge about them is limited. More conclusions and potential assets are analyzed in the chapter of Experiments and Discussion.

3.4 Chapter Summary

In this chapter we presented our methodology for efficient query routing and processing in peer-to-peer networks. Firstly we formulated the general problem that we were trying to solve and then we presented our design decisions for our proposed peer-to-peer network. After defining the basic query routing context we presented our proposed query routing and processing strategies and their significant advantages. We analyzed the use cases of our suggested top-k query processing algorithm according to the defined data scoring technique and present a cost analysis for it. Finally we presented an example of this proposed technique.

Chapter 4

Implementation

This chapter describes the prototype system HT-p2p. This system implements our proposed top-k query processing algorithm under a peer-to-peer network. The whole system is built upon the JXTA platform which provides a common set of open protocols and an open source reference implementation for developing general purpose, interoperable and large scale P2P applications. After presenting the basic features of JXTA we show in practice how HT-p2p algorithm can be used in general by any Super-Peer based peer-to-peer network.

4.1 JXTA Technology

4.1.1 Definition and Objectives

JXTA is an open network computing platform designed for peer-to-peer (P2P) computing [73, 74]. Its goal is to develop basic building blocks and services to enable innovative applications for peer groups. The term "JXTA" is short for *juxtapose*, as in side by side. It is a recognition that P2P is juxtaposed to client-server or Web-based computing, which is today's traditional distributed computing model. JXTA provides a common set of open protocols and an open source reference implementation for developing peer- to-peer applications. The JXTA protocols standardize the manner in which peers:

- Discover each other
- Self-organize into peer groups
- Advertise and discover network services
- Communicate with each other
- Monitor each other

The JXTA protocols are designed to be independent of programming languages, and independent of transport protocols. The protocols can be implemented in the Java programming language, C/C++, Perl, and numerous other languages. The

official website of JXTA [16] has Java and C implementations of the core protocols. They can be implemented on top of TCP/IP, HTTP, Bluetooth, HomePNA, or other transport protocols.

Project JXTA was originally conceived by Sun Microsystems, Inc. and designed with the participation of a small but growing number of experts from academic institutions and industry. It has a set of objectives that are derived from what we perceive as shortcomings of many peer-to-peer systems in existence or under development [73, 75]. Many peer-to-peer systems are built for delivering a single type of services. For example, Napster [1] provides music file sharing, Gnutella [2] provides generic file sharing, and AIM [76] provides instant messaging. Given the diverse characteristics of these services and the lack of a common underlying P2P infrastructure, each P2P software vendor tends to create incompatible systems - none of them able to interoperate with one another. This means each vendor creates its own P2P user community, duplicating efforts in creating software and system primitives commonly used by all P2P systems. Moreover, for a peer to participate in multiple communities organized by different P2P implementations, the peer must support multiple implementations, each for a distinct P2P system or community, and serve as the aggregation point. JXTA technology is designed to enable interconnected peers to easily locate each other, communicate with each other, participate in community-based activities, and offer services to each other seamlessly across different P2P systems and different communities.

Furthermore, the majority of current P2P systems offer their features or services through a set of APIs that are delivered on a particular operating system using a specific networking protocol. For example, one system might offer a set of C++ APIs, with the system initially running only on Windows, over TCP/IP, while another system offers a combination and C and Java APIs, running on a variety of UNIX systems, over TCP/IP but also requiring HTTP. It is obvious that approaches like the above are inefficient and impractical considering the dozens of P2P platforms in existence. JXTA technology is designed to be embraced by all developers, independent of preferred programming languages, development environments, or deployment platforms.

Also, many P2P systems, especially those being offered by upstart companies, tend to choose (perhaps unsurprisingly) Microsoft Windows as their target

deployment platform. The cited reason for this choice is to target the largest installed base and the fastest path to profit. The inevitable result is that many dependencies on Wintel-specific features are designed into (or just creep in) the system. This is often not the consequence of technical desire but of engineering reality with its tight schedules and limited resources. Definitely, this approach is clearly short-sighted, as P2P does not stand for PC-To-PC. Even though the earliest demonstration of P2P capabilities are on Wintel machines - the middle of the computing hardware spectrum, it is very likely that the greatest proliferation of P2P technology will occur at the two ends of the spectrum - large systems in the enterprise and consumer-oriented small systems. In fact, betting on any particular segment of the hardware or software system is not future proof. JXTA technology is designed to be implementable on every device with a digital heartbeat, including sensors, consumer electronics, PDAs, appliances, network routers, desktop computers, data-center servers, and storage systems.

To sum up JXTA provide a platform with the basic functions necessary for a P2P network supporting *interoperability*, *platform independenc*e and *ubiquity*. Project JXTA envisions a world where each peer, independent of software and hardware platform, can benefit and profit from being connected to millions of other peers.

4.1.2 JXTA Architecture and Protocols

The JXTA platform can be broken into three layers, as shown in Figure 16 below. Each layer builds on the capabilities of the layer below, adding functionality and behavioural complexity. At the bottom is the *core layer* provides the elements that are absolutely essential to every P2P solution. It deals with peer establishment, communication management and other low-level “plumbing”. Ideally, the elements of this layer are shared by all P2P solutions. In the middle is the *services layer* that deals with higher-level concepts, such as indexing, searching, and file sharing. The services layer provides network services that are desirable but not necessarily a part of every P2P solution. The *applications layer* builds on the capabilities of the services layer to provide the common P2P applications that we know, such as instant messaging, emailing, auctioning, and storage systems. Some features, such as security, manifest in all three layers and throughout a P2P system, albeit in different forms according to the location in the designed P2P software architecture.

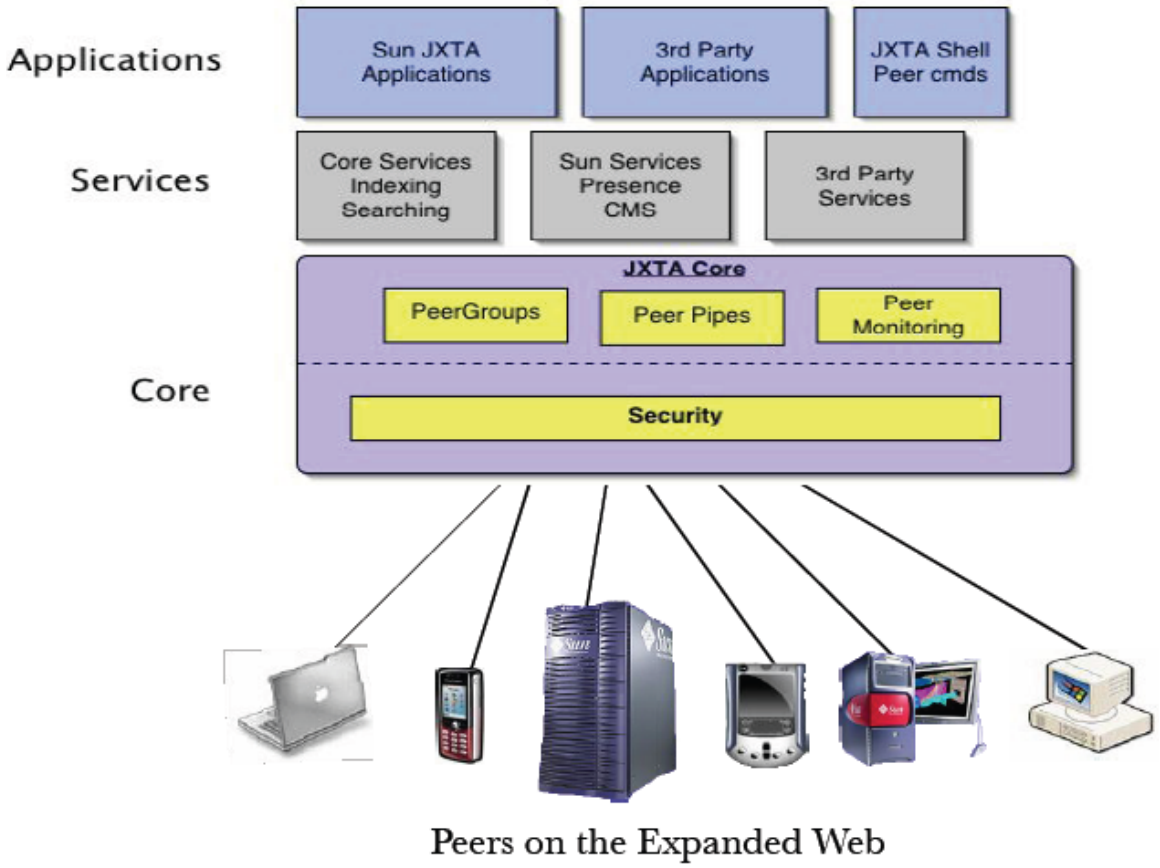

Figure 17: The JXTA three-layer architecture.

At the highest abstraction level, JXTA technology is a set of protocols:

- *Peer Resolver Protocol (PRP):* Used to send a query to any number of other peers and to receive a response.
- *Peer Discovery Protocol (PDP):* Used to advertise content and discover content.
- *Peer Information Protocol (PIP):* Used to obtain peer status information.
- *Pipe Binding Protocol (PBP):* Used to create a communication path between peers.
- *Peer Endpoint Protocol (PEP):* Used to find a route from one peer to another.
- *Rendezvous Protocol (RVP):* Used to propagate messages in the network.

Each protocol is defined by one or more messages exchanged among participants of the protocol. Each message has a pre-defined format, and may include various data fields. In fact, each of the JXTA protocols addresses exactly one fundamental aspect of P2P networking. Every protocol conversation is divided into a portion conducted by the local peer and another portion conducted by the remote peer. The local peer's half of the protocol is responsible for generating messages and sending them to the remote peer. The remote peer's half of the protocol is responsible for handling the incoming message and processing the message to perform a task. In Figure 17 below, the six different protocols are shown in their relationships to each other. The illustration further shows how a Java reference implementation can be built between the Java JRE and an application.

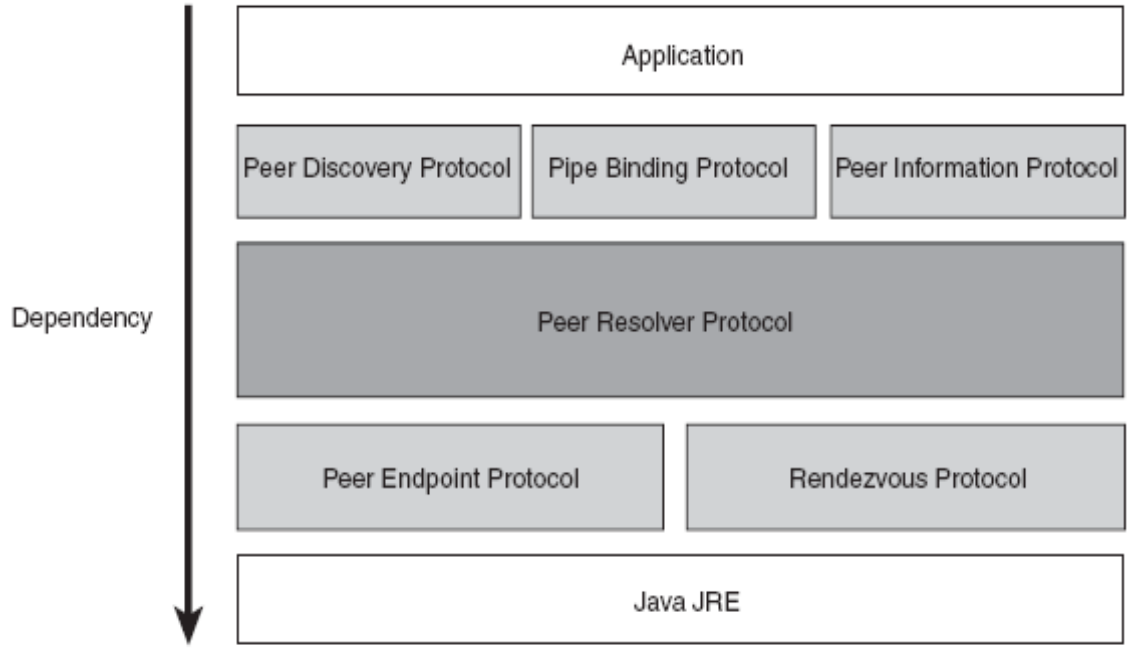

Figure 18: JXTA specification protocols hierarchy

4.1.3 JXTA Basic Concepts

4.1.3.1 Identifiers (IDs)

JXTA uses UUID, a 128-bit datum to refer to an entity. Currently, there are six types of JXTA entities which have JXTA ID types defined: peers, peer group, pipes, contents, module classes, and module specifications .The JXTA ID consists of three parts:

- Format specifier: urn
- Namespace identifier: jxta
- ID: unique value

It is important to note that the URN and JXTA portions of the ID are not case-sensitive, but the data portion of the ID is case-sensitive. An example of an ID is the following:`urn:jxta:uuid59616261646162614E50472050325033894 4BCED387C4A2BBD8E9415B78C48410`

4.1.3.2 Peers

The most common and widely understood component of any P2P system is the peer. A peer is simply an application, executing on a computer device, which has the ability to communicate with other peers. For the entire system to work, it is fundamental that the peer have the ability to communicate with other peers. For the purposes of JXTA, a peer is any networked device that implements the core JXTA protocols. This is the definition in the specification [73, 75], but we could note that a single "networked device" can have any number of JXTA peers executing on it. The peers could all be implementing different service code or participating in a computational complex algorithm. Each peer operates independently and asynchronously from all other peers, and is uniquely identified by a Peer ID. Moreover, JXTA introduces a special type of peer, the *Rendezvous peer* that is responsible for allowing a user to broadcast messages to other peers that belong to different local or private networks. These peers provide enhanced connectivity and contribute in avoiding message propagation to the entire network (message flooding).

4.1.3.3 Peer Groups

A peer group is a virtual entity that speaks the set of peer group protocols. Typically, a peer group is a collection of cooperating peers providing a common set of services. In general, peers self-organize into peer groups, each identified by a unique peer group ID. Each peer group can establish its own membership policy from open (anybody can join) to highly secure and protected (sufficient credentials are required to join). Peers may belong to more than one peer group simultaneously. By default, the first group that is instantiated is the *Net Peer Group.* All peers belong to the Net Peer Group. Peers may elect to join additional peer groups. The JXTA protocols

describe how peers may publish, discover, join, and monitor peer groups; they do not dictate when or why peer groups are created. A peer group provides a set of services called peer group services. JXTA defines a core set of peer group services. Additional services can be developed for delivering specific services. In order for two peers to interact via a service, they must both be part of the same peer group. The core peer group services include the following:

- *Discovery Service:* Allows searching for peer group content.
- *Membership Service:* Allows the creation of a secure peer group.
- *Access Service:* Permits validation of a peer.
- *Pipe Service:* Allows creation and use of pipes.
- *Resolver Service:* Allows queries and responses for peer services.
- *Monitoring Service:* Enables peers to monitor other peers and groups.

4.1.3.4 Advertisements

An advertisement is an XML-based document that describes and publishes the existence of a resource, such as a peer, a peer group, a pipe, or a service. Therefore, peers discover resources by searching for their corresponding advertisements, and may cache any discovered advertisements locally. All of the protocols use advertisements to pass information. JXTA technology defines a basic set of advertisements [75]. In addition, subtypes of advertisement can be formed from these basic types using XML schemas. The commonly used advertisement types are the following:

- *Peer Advertisement:* describes the peer resource by holding information about the peer, such as its name, peer ID, etc.
- *Peer Group Advertisement:* describes peer group-specific resources, such as name, peer group ID, description, specification, and service parameters.
- *Peer Info Advertisement:* describes the peer info resource by holding information about the current state of a peer, such as uptime, inbound and outbound message count, time last message received, and time last message sent.
- *Rendezvous Advertisement:* describes a peer that acts as a rendezvous peer for a given peer group.

- *Pipe Advertisement:* describes a pipe communication channel, and is used by the pipe service to create the associated input and output pipe endpoints. Each pipe advertisement contains an optional symbolic ID, a pipe type (point-to-point, propagate, secure, etc.) and a unique pipe ID.

An example of a pipe advertisement is presented at the following Figure 18.

```
<?xml version="1.0"?>

<!DOCTYPE jxta:PipeAdvertisement>

<jxta:PipeAdvertisement xmlns:jxta="http://jxta.org">
      <Id>
            urn:jxta:uuid-
59616261646162614E504720503250338E3E786229EA460DADC1A176B69B731504
      </Id>
      <Type>
            JxtaUnicast
      </Type>
      <Name>
            TestPipe.end1
      </Name>
</jxta:PipeAdvertisement>
```

Figure 19: An example of a pipe advertisement

4.1.3.5 Messages

The JXTA protocols are specified as a set of messages exchanged between peers. A message is an object that is sent between JXTA peers; it is the basic unit of data exchange between peers. Messages are sent and received by the *Pipe Service* and by the *Endpoint Service*. A message is an ordered sequence of named and typed contents called *Message Elements*. Thus a message is essentially a set of name/value pairs. There are two representations for messages: XML and binary. The data contained with in a MessageElement is accessible in four ways:

- as an InputStream
- ending the data as an OutputStream
- as a String
- as a byte array

4.1.3.6 Pipes

Pipes are asynchronous communication channels for sending and receiving messages. They are also uni-directional, so there are input pipes and output pipes. Moreover, pipes are virtual, in that a pipe's endpoint can be bound to one or more peer endpoints. Pipes are indiscriminate; they support the transfer of any object, including binary code, data strings, and Java technology-based objects. The pipe endpoints are referred to as the *input pipe* (the receiving end) and the *output pipe* (the sending end). Pipe endpoints are dynamically bound to peer endpoints at runtime. Peer endpoints correspond to available peer network interfaces (e.g., a TCP port and associated IP address) that can be used to send and receive message. JXTA pipes can have endpoints that are connected to different peers at different times, or may not be connected at all. Pipes offer two modes of communication, point-to-point and propagate. Thus we talk about two basic categories of JXTA pipes:

- *Point-to-Point Pipes:* A point-to-point pipe connects exactly two pipe endpoints together: an input pipe on one peer receives messages sent from the output pipe of another peer, it is also possible for multiple peers to bind to a single input pipe.
- *Propagate Pipes:* A propagate pipe connects one output pipe to multiple input pipes. Messages flow from the output pipe (the propagation source) into the input pipes. All propagation is done within the scope of a peer group. That is, the output pipe and all input pipes must belong to the same peer group.

Pipes on its general form are asynchronous, uni-directional, and unreliable, in order to gain the lowest overhead. But for many peer-to-peer applications there is a need of bidirectional and reliable communication channels. For these reasons JXTA provides two additional pipe types that built on top of pipes, endpoint messengers, and the JXTA reliability library which ensures message sequencing and delivery: *JxtaBiDiPipe/JxtaServerPipe* and JxtaSocket/JxtaServerSocket. In particular JxtaSocket and JxtaServerSocket are subclasses of java.net.Socket, and java.net.ServerSocket respectively. They provide stream based interface ala Socket, configurable internal buffering, and message chunking. However since they do not implement the Nagels algorithm [77], streams must be flushed as needed.

JxtaBiDiPipe and JxtaServerPipe provide a message based interface but provides no message chunking (applications need to ensure message size does not exceed the platform message size limitation of 64K). JxtaServerSocket, and JxtaServerPipe expose a input pipe to process connection requests, and negotiate communication parameters, whereby JxtaSocket, and JxtaBiDipipe bind to respectively to establish private dedicated pipes independent of the connection request pipe.

4.2 The HT-p2p system

4.2.1 System Description, Design Decisions and Basic Features

We should point out at first that our system description covers both the case of HT-p2p algorithm and HT-p2p+. In fact in order to support the case of HT-p2p+ we need some extra functionality at Super-Peers which is analyzed below. As its name denotes the HT-p2p system is a peer-to-peer system built upon a Super-Peer topology. A number of specified peers and Super-Peer(s) each time participate at the execution of HT-p2p algorithm in order to return the top-k objects to a given query. Each peer is assumed that it has a ranked list of its objects which defines pairs of the form (Object_id, Score). We suppose that each Object_id is unique and it is related with a specific object accordingly to one of the three use cases of HT-p2p as they presented in subsection 3.3.4.4. At each execution of HTp2p the participant peers and Super-Peer(s) are exchanging messages that include (Object_id, Score) pairs, thresholds and some control information such as their peer name.

For the organization and the implementation of the p2p network we chose to use *JXTA technology* which provides a number of objectives as we present them at the previous section. We selected to use the *Java* Version of JXTA (*jxta version 2.3.3*) in order to benefit from the application of both technologies. With Java our system is platform independent, with JXTA is network independent. Moreover since messages on JXTA platform are XML-based, we gain application independence by using this standard. XML supports interoperability across different applications, and in our case these applications could be different kind of peers (computers, PDAs, mobile phones etc). Furthermore for Java Version the JXTA community provides a lot of information (docs, forums, blogs, tutorials, source examples) and an open reference manual

contrary to the C-version. This is surely an extra advantage for all developers of p2p applications.

From JXTA Technology we used a variety of features in the implementation of HT-p2p system. First of all each peer and its corresponding peer group has a unique *JXTA ID*. Moreover we use JXTA *bidirectional pipes* for the communication between peers and Super-Peers and as such for the communication between Super-Peers. Of course each pipe has a unique JXTA ID (pipe id). All pipes are *Secure Unicast Pipe*s: a subcategory of *point-to-point pipes* that provides secure and reliable communication channels by supporting acknowledgement operations. Secure unicast pipes (JxtaUnicast) are classified as a derived type in [78]. Also the use of *pipe advertisement* is needed since our communication policy of peers is based on pipes. *Peer advertisements* are used for each peer who is joined into the p2p network and *peergroup advertisements* are used when a peer joins in a new peergroup. All messages that are transmitted across the network during the execution of HT-p2p are accessible as strings since their message elements are belong to *StringMessageElement* category.

Therefore from the above used technologies we can conclude that the HT-p2p system uses all the functionalities of the core level of JXTA architecture. It also implements the functionalities of the *Peer Discovery Protocol* (PDP) and the *Pipe Binding Protocol* (PBP) to build JXTA Services. Also the functionalities of *Rendezvous Protocol* (RVP) can be used for each participant Super-Peer.

Each JXTA peer runs on a unique port and to a specific IP Address. This information is described at a configuration file which is created the first time when we start the jxta platform for each peer. Specifically the first time a JXTA technology application is run, an auto-configuration tool (*JXTAConfigurator*) is displayed to configure the JXTAplatform for your network environment. This tool is used to specify configuration information for TCP/IP and HTTP, configure rendezvous and relay peers, and enter a user name and password. When the JXTA Configurator starts, it displays the Basic Settings panel (see Figure 19 below). Additional panels are displayed by selecting the tabs (Advanced, Rendezvous/Relay, and Security) at the top of the panel (see Figure 20 below).

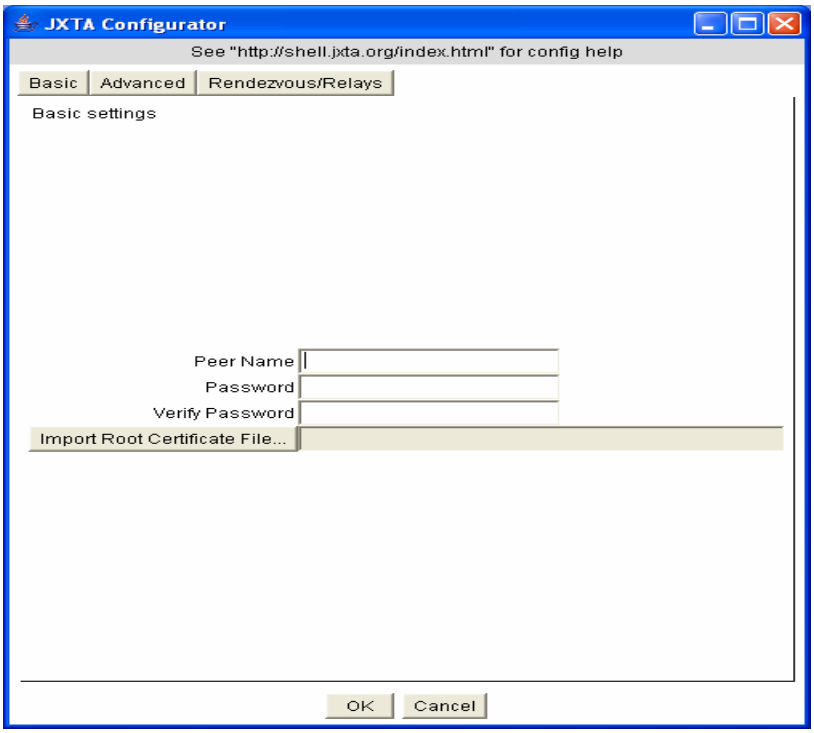

Figure 20: JXTA Configurator Basic Settings

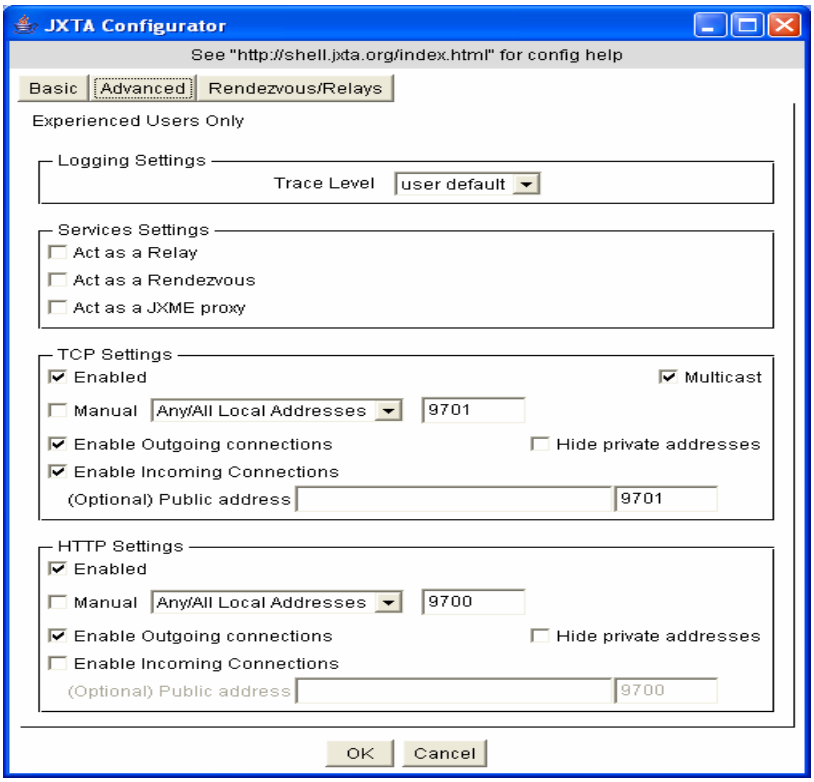

Figure 21: JXTA Configurator Advanced Settings

At HT-p2p we configure all peers to run using only the TCP Settings and a unique port for each IP address. By this way we could run more than one peers at the

same host computer as soon as they use a different port number and they are run at different command/terminal window.

4.2.2 System Design and Architecture

The HT-p2p's architecture consists of two modules according to the type of participant peer. Therefore we have the *Super-Peer Module* and the *Peer Module*. Each module communicates with each other using a third abstract module called *Communication Module* and denotes the communication policy between peers. In practise each module is implemented in Java as a different class which invokes its methods that are needed at each case of peer or Super-Peer. The Communication Module as an abstract module invokes the suitable methods both at the two cases. Figure 21 shows schematically the building blocks of HT-p2p's architecture.

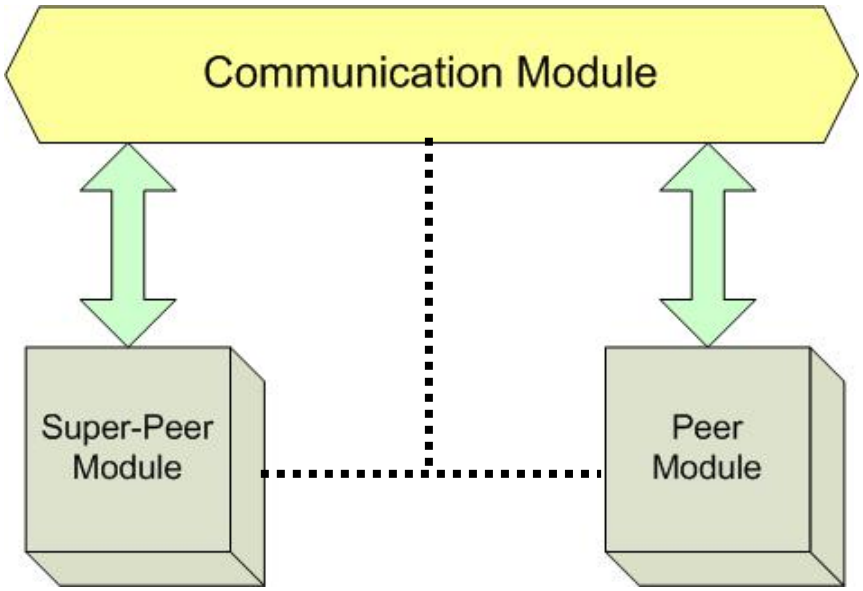

Figure 22: The building blocks of HT-p2p's architecture

This architecture can be applied for both two versions of our proposed top-k query algorithm. Therefore, each contributor peer implements Peer Module and each contributor (case of HT-p2p+) or collector (case of HT-p2p) Super-Peer the Super-Peer Module correspondingly. Each Super-Peer as we have assumed has a cluster of peers. This cluster can be a JXTA peergroup. At first when a new peer is joining into a peergroup it publishes its *PeerAdvertisement* on it. All peer advertisements are stored locally at a cache. The collector Super-Peer defines its role at the field

description of its corresponding advertisement. Thus when a Super-Peer looks for the collector Super-Peer it is needed to send a discovery message (or more than ones since the messages are sent asynchronously) in order to get the remote advertisements of Super-Peers to identify this one that denotes the role of collector.

As default option HT-p2p system we suggest that the originator Super-Peer to be the collector as well in order to skip the above process and to prevent sending one more message at the end of the algorithm where the collector Super-Peer sends to the originator Super-Peer the final top-k results. But this can be applied only in the case of HT-p2p, because if we have more than one running instances of the algorithm (case of HT-p2p+) constrainedly Super-Peers don't know which one is the collector. In the next subsections we analyzed in details each module of our proposed architecture

4.2.3 Communication Module

4.2.3.1 Implementation Decision

The communication module defines the communication policy of the whole p2p system. Since we need bidirectional and reliable communication channels between connected peers and Super-Peers we should decide if we choose to use JxtaBiDiPipes or JxtaSockets in order to establish each required communication channel. We finally chose *JxtaBidiPipes* and *JxtaServerPipes* to implement the communication policy of HT-p2p instead of using JxtaSockets and JxtaServerSockets respectively. Our decision first of all is made by taking into account the functionality and flexibility of message-based interface that provides JxtaBidiPipes instead of stream-based interface of JxtaSockets. According to HT-p2p algorithm each peer sends at each phase the required data independently from other peers. Obviously, it is easier to do this by sending a message instead of writing the data into a socket. In the former case the Super-Peer just receives messages, while in the last case it should process the writing socket and its corresponding stream each time.

In [79] there is an evaluation of JXTA Communication Layers. It was come from the benchmarking of each communication layer of JXTA-J2SE as it is available via the web site of JDF [80] project. This evaluation showed that the throughput difference between JXTA sockets and JXTA pipes for sending large messages is negligible. But on the latency side JXTA Pipes gained better times than JXTA

Sockets. Also in the experimental test of [81] JXTA Pipes seemed to be a little more efficient than JXTA Sockets. Thus we conclude that our decision for the use of JXTA Pipes for our peer-to-peer scenario is surely correct.

4.2.3.2 Basic Functionality

The communication module defines the communication protocol between peers and Super-Peers. Each Super-Peer of the p2p network is required to make a *PipeAdvertisement*. This advertisement is known to all members of the specific group of peers (cluster) that is responsible the specific Super-Peer. It has a unique JXTA ID and it defines the type of the used pipe as communication channel. Each Super-Peer makes a *JxtaServerPipe* for a specific *PeerGroup* with the specific above PipeAdvertisement. At this all the participant (contributor) peers are connected by using a *JxtaBidiPipe*. There is a specified timeout for each peer to connect to the server pipe. At HT-p2p we adjust to 18000 milliseconds. As long as the connection has been established through a pipe, peers are ready to send and receive messages across this pipe (see Figure 22 below).

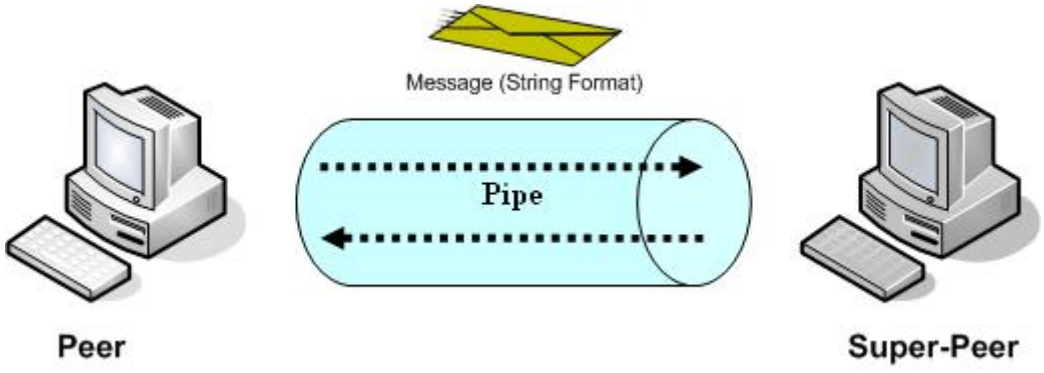

Figure 23: Bidirectional connection through pipe for message transfer

The communication module defines the corresponding methods for sending and receiving messages. Each peer sends a message using a tag. In order to receive another peer the message from the former peer it needs to have the same tag which it is used to retrieve each message from the message queue. This tag is defined as a namespace at the JXTA API and it has string representation. Moreover peers by using a *PipeMsgListener* receive messages through the method *pipeMsgEvent* which exploits a message event to get each incoming message at a specific pipe. Super-Peers use one tag to communicate with a specific peer group of peers. In the case of HT-

p2p+ where is needed the communication between Super-Peers they use another tag for sending/receiving messages through a different pipe. In particular, in the last case a new *JxtaServerPipe* is required to bind all the JxtaBidiPipes from the connector Super-Peers.

4.2.4 Super-Peer Module

HT-p2p system is designed to execute in a distributed way each phase of the algorithm. Therefore, since we want each Super-Peer to communicate and to process each peer's request independently at each phase, we decided to implement a multi-threaded architecture for the Super-Peer Module. In general, the running of multiple threads in an application at the same time performs different tasks at this application [82, 83]. Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority. But as long as peers are equal, we adjust all threads to have the same priority in order to be processed equally from the specified Super-Peer. Thus if for example (Figure 23) peer1 sends their (Object_id, Score) pairs in a request and peer2 its own pairs in another request, then the Super-Peer will process them independently. Therefore, at the same time and it could send back some results in parallel to each peer, since for each request a new thread is made to serve it.

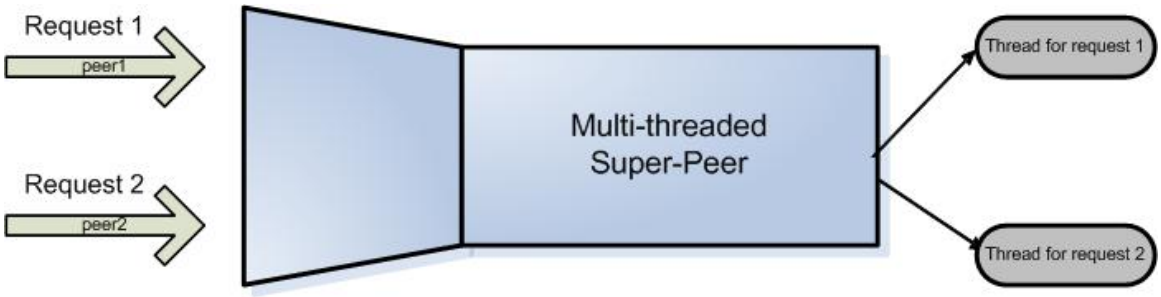

Figure 24: Multi-Threaded Architecture of Super-Peer

After Super-Peer has started the JXTA platform by calling the corresponding method (startJxta) and has applied its communication policy with its contributor peers, it starts running the execution of the HT-p2p algorithm. For each phase of the algorithm at least one separate method is called. Thus the Super-Peer module invokes the following methods for each phase:

- **Phase 1:** *superPeerPhase1*→ Determines phase1 bottom and returns the k objects with the k-highest partial sums till now.
- **Phase 2**: *superPeerPhase2a* → sends the list with the top-k object IDs from the previous phase and the threshold T to the contributor peers and receives from them their suitable (Object_id, Score) pairs.
- **Phase 2**: *superPeerPhase2b* → finds out the objects with the k-highest partial sums, determines phase2 bottom, and checks the condition to proceed to next phase or to return the results.
- **Phase 3**: *superPeerPhase3a* → sends Tpatch to all required peers, receives their suitable pairs and calls superPeerPhase3b.
- **Phase 3**: *superPeerPhase3b* → Checks if objects from previous call are contained at the set of seen objects, determines phase2 bottom, calculates upper bounds for all seen objects, and calls superPeerPhase3c.
- **Phase 3**: *superPeerPhase3c* → removes dismissed objects from current set of seen objects calls superPeerPhase4 and returns the top-k candidate set of objects.
- **Phase 4**: *superPeerPhase4* → calculates the real scores for the objects than belong to top-k candidate set and return the final top-k results.

Except from the above methods each Super-Peer as implements the Super-Peer Module invokes additionally a set of methods that are required during the real execution of HT-p2p and are called inside of these basic ones. We report the rest methods and we describe briefly its role:

- *findMaxSums*→ calculates the k highest partial sums and returns them.
- *messageParsing* → parses each message according to its content at each phase.
- *calculatePartialSum* → calculates a partial sum for a given object
- *calculateUpperBound* → calculate upper bounds for all seen objects and return the objects whose upper bound is less than phase 3 bottom (called dismissed objects).

- *findSeenObjects* → finds out which objects has been reported by the contributor peers (called seen objects).
- *processSums* → manipulates all partial sums and process them according to the running phase of the algorithm.
- *removeDuplicateObjects* → removes duplicate objects from a given set (i.e. seen objects).
- *saveAndReportObjects* → stores in a hashtable its contributor's peer name and its reported (Object_id, Score) pairs.
- *saveThresholds* → stores all the requires thresholds during the execution of HT-p2p.

At the Super-Peer Module the corresponding role for each Super is defined using some flags (ORIGINATOR, COLLECTOR). In the case of HT-p2p+ we need an extra flag (CONTRIBUTOR) to indicate whether this Super-Peer is running an instance of the algorithm or not. Certainly, we need some extra functionality to support the combination of results from different Contributor Super-Peers at Collector Super-Peer. Specifically after the execution of the algorithm has finished, the method *processHTp2pResults* is called. This method if the specific Super-Peer has not the role of Collector then it calls another method called *findCollectorAdv*. This method sends a *discovery message* in order to find the Collector Super-Peer and retrieve its PipeAdvertisement. Afterwards this specific Super-Peer calls *readCollectorAdv* in which reads the Collector's PipeAdvertisement and uses it to connect to its JxtaServerPipe. Moreover by calling the *readCollectorAdv* method the specific Super-Peer undertakes to send through this pipe its top-k objects to the collector Super-Peer. The last by calling the method *extraPhase* accumulates all the discrete candidate objects and sums their real scores. For the calculation of real scores extraPhase examines if all scores from candidate objects has been sent by all participant contributor peers. If not, it requests from peers to send them in order to be the calculation of real score completed and correct. Finally *extraPhase* returns the objects with the k-highest aggregated real scores as top-k objects. To sum up for the case of HT-p2p+ the following methods are invoked in the Super-Peer Module:

- processHTp2pResults

- findCollectorAdv
- readCollectorAdv
- extraPhase

4.2.5 Peer Module

The Peer Module is surely simpler than Super-Peer Module. Each contributor peer has to score its relevant objects before start to participate to the algorithm as a contributor peer. This is done through a method at HT-p2p system called *scoresLoading*. After the call of the scoresLoading method and the starting of JXTA platform through the call of *startJxta* method and the application of communication policy, each peer start to execute its own steps of the HT-p2p algorithm as soon as it has connected to the suitable pipe of the contributor/collector Super-Peer. We should note that the Peer Module uses *PipeMsgEvent* from the Communication Module to get each incoming message and process it for each phase of the algorithm. For each phase the basic invoked functions in Peer Module are the following:

- **Phase 1**: *peerPhase1* → sends top-k objects and their scores to the specified Super-Peer.
- **Phase 2**: *peerPhase2a* → receives top-k object IDs in a list L and threshold T from Super-Peer and finally returns its threshold Ti by determining its lowest score among all the k objects in the list and compare it with the threshold T.
- **Phase 2**: *peerPhase2b* → sends the (Object_id, Score) pairs where Ti < Score.
- **Phase 3**: *peerPhase2b* → sends the (Object_id, Score) pairs where Tpatch > Score.

We have to point out that at Phase2 and Phase 3 the same method is called with different parameter to process a similar but different task. Finally, at Peer Module it is invoked one more method called *messageParsing* which parses each message according to its content where is needed at each phase of the peer.

4.3 Chapter Summary

In this chapter we described the prototype system HT-p2p that implements our suggested top-k query processing strategy built upon JXTA platform. After presenting the basic features of JXTA technology we made a description of the whole system and its constitutional modules. Thus we showed in practise how HT-p2p algorithm can be used by any Super-Peer based peer-to-peer network.

Chapter 5

Evaluation

In this chapter we present a set of experiments that we performed to evaluate the HT-p2p algorithm. Our experiments were focused on the characteristics of HT-p2p under our defined peer to peer scenario. Thus we carried out some different tests on our implemented on JXTA platform system (HT-p2p System) We measured not only the performance of HT-p2p in its standard phases, but also in its extra phase where more than one Super-Peer is needed to contribute in order to get back the results (case of HT-p2p+).

5.1 Experimental Setup

First of all we have to point out that our measurements do not to intend to check the speed of HT-p2p algorithm, because this should come from a simulation framework with million participant peers, but to test it under real conditions and with different parameters in order to evaluate its basic characteristics that contribute to scalability and efficiency.

For Super-Peers we assumed that they are online and they have created (read) their PipeAdvertisement. For peers we also assume that they are online, they have read the PipeAdvertisement of each corresponding Super-Peer so they joined in each peer group. This assumption is necessary in order to start measuring the clear execution time of HT-p2p without taking into account the time that the peers need to join into the peer-to-peer network. We used the same scoring function for each peer which returned a random score for each object which belonged to specific range of values.

The scoring function's range was selected such that the peers in each running of HT-p2p will send a big number of objects comparatively to the number of the objects that are stored in each database. This was done in order to evaluate HT-p2p under "difficult" conditions of our experiments and not under conditions that HT-p2p runs fast, processes few objects, without any remarkable differences in performance of HT-p2p. We have to mention that the scoring of objects was done before the start

of HT-p2p. For each experiment we report the hardware and software which was used to perform the test. For all experiments all computers are connected through LAN of 100 Mbit/sec.

5.2 Experiments

5.2.1 Experiment 1

Hardware Used for Super-Peer: Pentium4 3.4 GHZ, 2.096 MB RAM
Hardware Used for Peers: Pentium4 2.8 GHZ 1.047 MB RAM
Operating System: Windows 2000 Professional
Software Used: Eclipse Version 3.0 [84], Java Version "1.5.0_04"

The first experiment was done to evaluate HT-p2p's behaviour under different values of k. Also in parallel, we examine how the running of Patch Phase (Phase 3 of the algorithm) affects the execution of HT-p2p, as the value of k increases. In this experiment all peers were run at one computer while Super-Peer was run at a different host computer. Each peer in this experiment has in its database 150 ranked objects. The results of this experiment are shown in Table 4 above.

Super-Peer (s)	Contributor peers	Patch Phase(s)	k	Average Execution Time (ms)
1	9	0	**k = 5**	7836
1	9	1	**k = 5**	5366
1	9	2	**k = 5**	4920
1	9	0	**k = 10**	9489
1	9	1	**k = 10**	10203
1	9	2	**k = 10**	9222
1	9	0	**k = 25**	8990
1	9	1	**k = 25**	8040
1	9	2	**k = 25**	8730
1	9	3	**k = 25**	8753
1	9	0	**k = 50**	9082
1	9	1	**k = 50**	9300
1	9	2	**k = 50**	9333
1	9	0	**k = 100**	11194
1	9	1	**k = 100**	8716
1	9	0	**k = 125**	12831
1	9	1	**k = 125**	10204
1	9	2	**k = 125**	8571

Table 4: Results Table of Experiment 1

We run each case at least two times and we observed fluctuations on the execution time. For this reason we got average execution times in milliseconds. The fluctuations are explained because we use a random scoring function and some objects are taking higher or lower values in some runs and in different peers. Thus there were some cases where HT-p2p has to process more or less seen objects. Although, from the above results we could make the conclusion that the value of k does not really affect the execution time of HT-p2p. Examining the case where we didn't have any patch phase (Figure 24 below) and the value of k was 10, 25, 50 and all the execution times are almost the same (approximately 9 sec). Also, when k = 5 we just observed a little less time (approximately 8 sec) which is the simplest case. A more noticeable but small as well aberration for the same case was observed when k was greater than 100 where the execution time was greater than 11 sec. But when k >=100 and at least one patch phase is run this aberration seems to disappear. This has to do with our next observation.

So, another important observation is that the Patch Phase in general doesn't endorse the overall performance of HT-p2p. We strengthen this conclusion provided we note at the above table that there are runs where the existence of Patch Phase not only increases execution time, nor decreases it. This can be explained by the fact that in Patch Phase we have a pruning of top-k candidate set. Thus, although we need more time to send some extra objects to Super-Peer which has to process them, at the last phase since the top-k candidate set will be smaller, the calculation of final scores will take less time.

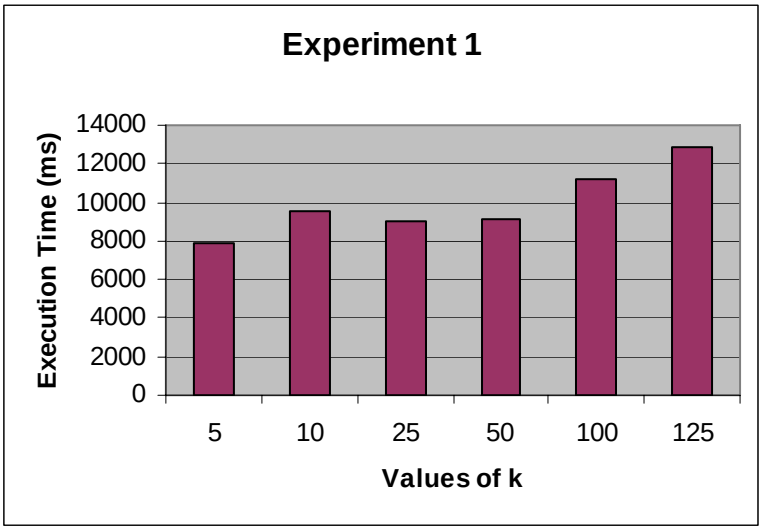

Figure 25: Execution time as k increases in HT-p2p

5.2.2 Experiment 2

Hardware Used for Super-Peer: Pentium4 3.4 GHZ, 2.096 MB RAM
Hardware Used for Peers: Pentium4 2.8 GHZ 1.047 MB RAM
Operating System: Windows 2000 Professional
Software Used: Eclipse Version 3.0 [84], Java Version "1.5.0_04"

The second experimental test was done to evaluate the performance of HT using an increasing number of peers that contribute to the query results (contributor peers). Since the number of messages that are transmitted during the execution of the algorithm are fixed we measure the execution time of HT-p2p. Our goal was to observe how the algorithm is affected by the number of the contributor peers. In other words we tested the scalability of the algorithm. Also in parallel, we examine whether our claim about the Patch Phase take affect under the execution of HT-p2p, as the number of contributor peers increases.

We assumed that each peer has in its database 150 ranked objects. This Super-Peer has the role of the Originator Super-Peer and Collector Super-Peer as well, so the specific Super-Peer returned the results of the specific query that was submitted to HT-p2p system. Our Super-Peer was run on one computer and peers were run on one another computer. Since the value of k doesn't affect the execution time (as we saw at the previous experiment), all runs in this experiment are made with k = 10. The results are shown on the following Table 5 for this experiment called Experiment 2a.

Contributor peers	Patch Phase(s)	k	Execution Times (ms)	Average Time (ms)
2	0	10	231,250,290, 360	*283*
2	1	10	250,301,381, 395	*332*
2	2	10	281,381,411,450	*339*
4	0	10	501,700,741,892	*708*
4	1	10	561,651,731,951	*723*
4	2	10	871,881,931,941	*906*
6	0	10	1408, 2082,3561,4586	*2908*
6	1	10	1422 , 2703,3945,4196	*3067*
6	2	10	1533, 2801,4003,4234	*3143*
8	0	10	4743,5828,7421,8541	*6633*
8	1	10	4531, 6203, 7589, 8903	*6806*
8	2	10	4663, 4997, 8498, 8899	*6764*
10	0	10	4956,9863,10313,10624	*8939*
10	1	10	5013, 8947, 9923, 10627	*8628*
10	2	10	4733, 8053,9923,11112	*8455*

12	0	10	7981, 13678, 14759, 21338	*14394*
12	1	10	9903, 11213, 16556, 20456	*14532*
12	2	10	7733, 10956, 16889, 22005	*14395*

Table 5: Results Table of Experiment 2a

We run each case four times and we observe fluctuations on the execution time for same cases. Hence we took into account the average execution time of the algorithm. One explanation for this fact could be the use of our random scoring function (Experiment 2a) that in some runs there are objects are taking higher or lower values in different peers. Thus, there are some cases where HT-p2p has to process more or less seen objects. Although, from the results of Experiment 2a (see related Table 5 above) we could make the conclusion that as the number of contributor peer increases the execution time of HT-p2p is increasing too. From the next Figure which depicts the case when there was not any patch phase we can conclude that the increase of execution time is linear.

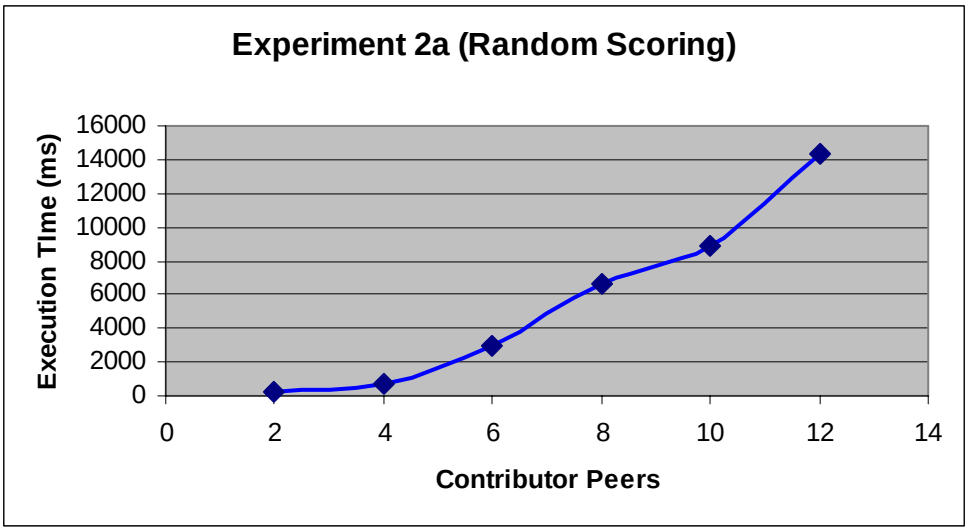

Figure 26: Execution time as contributor peers are increased in HT-p2p (use of random scoring function)

In order to have a more complete view on how this increase fluctuates as the number of peers grows we proceeded to Experiment 2b. In this experiment we used a standard scoring function in order to observe the affection of peer's growth to the algorithm performance where all the peers process the same number of objects at each phase of the algorithm. The results for Experiment 2b are shown in Table 6 below and the corresponding chart for the case when there was not any patch phase is shown in Figure 26. We run this experiment at least 3 times for each case.

Contributor peers	Patch Phase(s)	k	Execution Times (ms)	Average Time (ms)
2	0	10	181, 220, 220	*207*
4	0	10	461, 481, 641	*527*
6	0	10	591, 731, 972	*764*
8	0	10	1222, 1542, 1922	*1562*
10	0	10	1162, 1402, 1775, 2564	*1725*
12	0	10	1342, 2035, 2593, 3754	*2431*

Table 6: Results Table of Experiment 2b

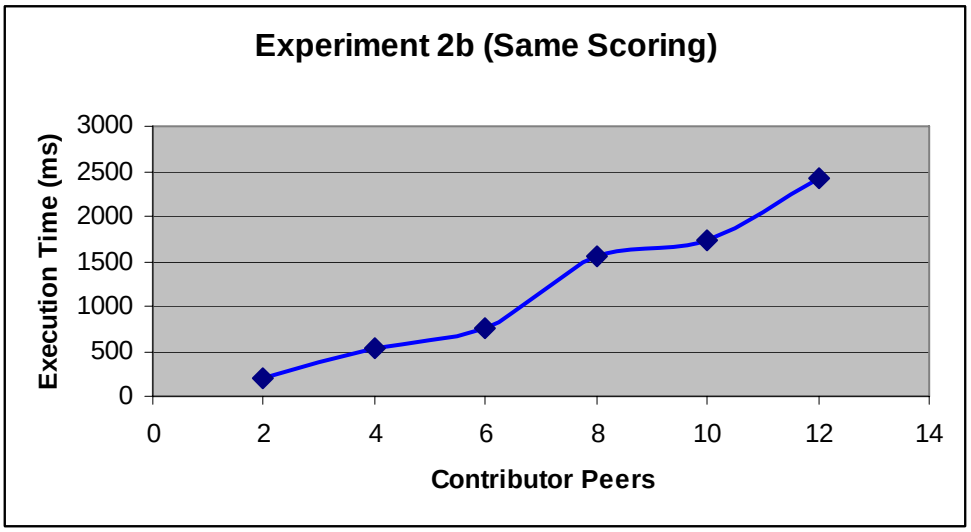

Figure 27: Execution time as contributor peers are increased in HT-p2p (use of same scoring function)

Although in this experiment the fluctuations for same cases were surely shorter than these at Experiment 2a we observed different percentage increases by doubling each time the number of peers. We conclude that the explanation for all these fluctuations in the same cases is due to the distributed running of phases for each peer. In some cases peers are running in a full distributed way, that's why we get smaller execution times. This has to do with the running of phase 2 at Super-Peer where the last is waiting for all peers to finish their phase 1 until it defines the phase2 bottom and broadcasts the suitable thresholds to them. Though, the general result of Experiment 2 is the linear increase of execution time as the number of contributor peers increases.

Another important observation is that our claim about the Patch Phase becomes true. We conclude that in general the Patch Phase doesn't endorse the overall

performance of HT-p2p. Thus there also are runs of this experiment where the existence of Patch Phase not only increases execution time, nor decreases it.

5.2.3 Experiment 3

Hardware Used for Super-Peer: Pentium4 3.4 GHZ, 2.096 MB RAM
Hardware Used for Peers: Pentium4 2.8 GHZ 1.047 MB RAM
Hardware Used for Collector Super-Peer: Pentium4 3.4 GHZ 2096 MB RAM
Hardware Used for contributor Super-Peers: Pentium4 3.4 GHZ 2096 MB RAM
Operating System: Windows 2000 Professional
Software Used: Eclipse Version 3.0, Java Version "1.5.0_04"

In the case of HT-p2p+ the contributor Super-Peers are sending their top-k results to the collector Super-Peer who undertakes to combine all top-k results and at last returns the final top-k results. This process is supported by some extra methods (as we have seen at the implementation of the HT-p2p system, see previous Chapter for details) which takes surely some time to execute. In this experiment we measure how these extra methods affect HT-p2p+ performance.

We run some *top-10 queries* where the number of Super-Peers was grown at each running and the number of contributor peers for each Super-Peer was fixed. All peers were run on the same computer while Super-Peers were run on separate host computer. We assumed that for the calculation of real scores we didn't need extra contribution from peers for sending scores of unseen objects. For each case we observed that extra phase takes just *10 ms* to the collector Super-Peer for each running of it. Also for each contributor Super-Peer one additional message (discovery message) is needed in order to find the collector Super-Peer at the peer-to-peer network. This message takes *2040-2100 ms*. Also, each contributor Super-Peer needed about *340-401 ms* to read collector's advertisement in order to connect to its pipe and sends its top-k object set. To sum-up the results for each method of HT-p2p+ were the following:

- findCollectorAdv at each contributor Super-Peer: *2040-2100 ms*
- readCollectorAdv at each contributor Super-Peer: *340-401 ms*
- extraPhase at Collector: *10 ms* for this case

In the worst case where Super-Peers are not run in a distributed way we need *2100 + 401 = 2501 ms* for each contributor Super-Peer and *10 ms* for the collector Super-Peer for this kind of query at HT-p2p+ system.

It has practical meaning how this communication cost of this discovery message endorses the performance of HT-p2p if the Super-Peers are not in the same local subnet network. We observe a delay of *102-508 m*s for each contributor Super-Peer in order the collector Super-Peer to receive the top-k results from them.

In general we conclude that these extra methods don't endorse to large extent the HT-p2p's performance. For this experiment, we observed that the most time-consuming operation is the detection of collector Super-Peer (in method findCollectorAdv) which is achieved through a discovery message to the Super-Peer backbone. Surely the total cost also depends on the combination of top-k object sets (in method extraPhase) which in turn depends on the number of unseen objects in each contributor Super-Peer.

5.3 Chapter Summary

In this chapter we presented our performed experiments that were based on the characteristics of HT-p2p algorithm. For each experiment we described the hardware and software which we used. After denoting the experimental setup we presented each experiment and its results separately. We took into account both the cases of HT-p2p and HT-p2p+ algorithm.

Chapter 6

Conclusions

The general conclusions of this thesis are presented in this chapter. In particular we summarize our work and report briefly its contributions. We also provide some extensibility suggestions to the query routing strategy and to the query processing strategy as well. Moreover we mention some open issues for future work.

6.1 Summary

Peer-to-peer networks can be seen as the alternative proposed to overcome the limitations of client-server model. Thus peer-to-peer systems were designed in order to take advantage of resources at the edges of the network and to promote the sharing of these resources. There are many categories of peer-to-peer systems; each of them is applied for different cases of use. The advent of Semantic Web gave rise to a new category of peer-to-peer systems called Schema-Based. In Schema-Based P2P systems each peer is a whole database management system in itself. Each peer can use its own database schema, manages its own data and by this way maintains its autonomy. The combination of Semantic Web and Peer-to-Peer technologies results in building large scale peer-to-peer systems that support knowledge reuse and can provide effective solutions for searching across a p2p network.

Considering a Schema-Based peer-to-peer network our main goal is the easy sharing of knowledge bases which implies efficient exchange of data across the p2p network. In practise our goal would be achieved if each query is not broadcast into the whole network, but is routed only to relevant peers. The last formulates the problem of query routing in peer-to-peer networks. In this work we present the current dominant approaches in order to denote the state of the art for this problem. Furthermore, we suggest our solution to the problem of query routing. Our suggested routing strategy is based on an unstructured Super-Peer based architecture and on a well defined query routing context. It utilizes that each peer has its own RDF/S schema which describes its database schema information including a taxonomy of

terms that is related with the specific database and other user-defined relationships and properties. The clustering of peers according to semantic information by using the notion of Semantic Overlay Networks contributes to the main goal of our suggested routing strategy: the query is routed to suitable peers from corresponding super-peers in such way to avoid network traffic and bandwidth consumption.

Going one step ahead, the efficiency and the good performance of the whole peer-to-peer network depend not only on how the query is routed to relevant peers, but also on how it is processed by these relevant peers. This formulates the problem of query processing which is the next step after the query routing task. Generally query processing is dependent from the query routing strategy. A better performance can be achieved if both of them can be cooperated together in a smart way. Query processing in peer-to-peer networks is a multidimensional topic that many authors have worked on it suggesting a variety of techniques according to the hypotheses and the aspect of the problem each author formulates. But, for Schema-Based peer to peer networks there is a minimal work in query processing of such systems that is based on query planning and optimization and is presented briefly.

The new trend in query processing is the adaptation of top-k retrieval algorithms in order to get back the results quickly and without any large processing cost. This technique has just started to apply for distributed environments. We present the approaches of this direction and mention its corresponding advantages and drawbacks. Two of them seem to be the most suitable for large-scale distributed networks. We finally conclude that the Hybrid Threshold (HT) algorithm could be the best solution for top-k processing in peer-to-peer networks. We extend HT and adapt it under our well defined peer-to-peer environment and in consequence we suggest two improved versions: HT-p2p and HT-p2p+. The first assumes that results are returned by executing an instance of the algorithm to a specified Super-Peer named in this case collector Super-Peer. The last assumes that results come from the combination of all top-k object set that are returned from each running instance of the algorithm to each specified contributor Super-Peer. In addition, since HT-p2p belongs to score-based top-k algorithms, we study the problem of scoring objects and we suggest accordingly three use cases of the algorithm. In order to evaluate HT-p2p and HT-p2p+ we implemented a prototype system called “the HT-p2p system”.

The HT-p2p system is built upon JXTA platform which provides a common set of open protocols and an open source reference implementation for developing general purpose, interoperable and large scale P2P applications. In HT-p2p system our suggested top-k processing strategy is implemented under a simple and scalable architecture. HT-p2p utilizes the basic features of JXTA technology in order to provide a functional, manageable and efficient solution. In this way we show in practise how HT-p2p algorithm can be used in general by any Super-Peer based peer-to-peer network.

Finally we use HT-p2p system to make some experiments in order to evaluate our proposed top-k query processing technique. Our experiments focus on the characteristics of the algorithm. The results showed that HT-p2p does not affected by the value of k, so it can process with the same efficiency queries with small or big value of k compared to the number of stored and scored database objects. Also, HT-p2p has good scalability since as the number of contributor peers the execution time of it increases linearly. We also conclude that the Patch Phase of the algorithm does not in general endorse the overall performance due to the pruning of top-k candidate set. Moreover in the case of HT-p2p+ we measured the extra processing cost at each Super-Peer. Finally we observed different execution times for same cases which have to do with the distributed way of running. The last is related with the phase2 of the algorithm where the collector Super-Peer is waiting for all contributor peers to finish their phase 1 in order to define the phase2 bottom and broadcasts the suitable thresholds to them.

Consequently, after surveying the existing dominant approaches, this work suggests a complete framework for efficient query routing and processing in a Schema-Based P2P network. This framework by its suggested query routing and processing techniques supports fast query answering, easy data sharing, stability, privacy, self organizing, autonomy and load-balancing without flooding the network with aimless messages that waste probably the most important thing in a large scale network: its bandwidth.

In the next sections we present some extensibility suggestions on our query routing and processing techniques. Some of them are directly applicable and some other need some form of adaptation in order to fit in our defined peer-to-peer environment.

6.2 Extensibility Suggestions

6.2.1 Suggestions for Query Routing Strategy

Our proposed query routing strategy can be benefit if we decide to apply some *caching mechanisms*. It is meaningful to keep information from previous queries at each cluster of peers. Specifically each Super-Peer could cache path expressions of each query and could keep the relevant peers to the query as they have returned from the application of the routing algorithm. In this way if the schema navigation of the query has been processed at the past then we could look at a stored set of peers to find for relevant objects. The latter would makes our query routing technique surely faster. But we should denote in this case the applied cache replacement strategy. The most known replacement strategies are FIFO (First In First Out), Random, LRU (Least Recently Used) and LFU (Least Frequently Used). More sophisticated caching mechanisms such as the ones that are found in [85, 86] can be supported with the suitable adaptations to our peer-to-peer environment.

Furthermore we can employ an even more flexible the definition of relevant peers by using external schemes such as *Wordnet* [87, 88] for the matching of required terms with the query predicates. Wordnet is a lexical reference system which contains English nouns, verbs, adjectives and adverbs organized into synonym sets, each representing one underlying lexical concept. Different relations link the synonym sets. Thus we can look for a synonym of the query predicate that could match with at least one term of the specified taxonomy in order to determine some relevant peers.

In our routing technique which is based on Super-Peer topology of the peer-to-peer network, queries are routed through their responsible Super-Peers which we have assumed that they are always online. But if one participant Super-Peer fails, its clustered peers are temporally at a loose end. To provide reliability to the cluster and decrease the load on the Super-Peer we could use the notion of *k-redundant Super-Pee*r which was introduced in [66]. A super-peer is k-redundant if there are k nodes sharing the super-peer load, forming a single “virtual” super-peer. Every peer in the virtual super-peer is a partner with equal responsibilities: each partner is connected to every client and has a full index of the clients’ data, as well as the data of other partners. Peers send queries to each partner in a round-robin fashion; similarly, incoming queries from neighbours are distributed across partners equally. Hence, the

query load on each partner is a factor of k less than on a single Super-Peer with no redundancy. Therefore, a k-redundant Super-Peer has much greater availability and reliability than a single super-peer and could be used to solve the specific problem of failure.

6.2.2 Suggestions for Query Processing Strategy

For query processing strategy our suggestions are mainly based on the improvement and optimization of HT-p2p algorithm. As we observed from the use of HT-p2p system and its evaluation, in a few cases the collector super-peer needs to resend (Object, Score) pairs that have been considered at previous phases and their partial sums have been calculated. For optimization reasons we could apply some extra controls for the comparison of required (Object, Score) pairs with these ones that have been stored at previous sections by their corresponding responsible Super-Peers. From this comparison we could determine which exactly (Object, Score) pairs should be sent at each phase.

Furthermore, in order to overcome the involuntary delay of Phase2 (as it was affirmed from our experiments) of the algorithm we could suggest a specified timeout limit for each peer, in order to finish its phase1. If a peer could not overtake to finish its phase1 then its responsible Super-Peer should exclude it from the set of relevant participant peers at the specific running instance of HT-p2p. Surely we could put timeout limits as well to the other phases (phase 2, phase 3, phase 4), but in this case we should provide a recovery mechanism in order to delete stored results at Super-Peers for the excluded peers and to recalculate the required partial sums.

Finally, the support of top-k join queries is meaningful and should be applied to peer-to-peer networks as part of the query processing technique in the future. There is only a minimal work of supporting top-k join queries in relational databases [54, 55, 56]. The approach of [54] introduces some ideas that could be applied for peer-to-peer systems under some circumstances and with the suitable adaptations that need to be defined. Therefore, in general the problem of supporting top-k join queries is an open research topic for large scale distributed systems like peer-to-peer systems.

6.3 Chapter Summary

In this chapter we made at first a summary of the whole work. Thus we can easily conclude its contributions which we reported briefly. In addition we provided some extensibility suggestions to the query routing and processing strategy as well. Some of them are directly applicable and some other need some form of adaptation in order to fit in our defined peer-to-peer environment. Also, we mentioned some open issues for future work.

Chapter 7

References

1. The Napster file-sharing system. http://www.napster.com.
2. The Gnutella file-sharing system. http://www.gnutella.com.
3. Clarke, O. Sandberg, B. Wiley, and T.W. Hong. Freenet. *A Distributed Anonymous Information Storage and Retrieval System*. In Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability, volume 2009. Springer Verlag, 2001.
4. The Morpheus file-sharing system. http://www.morpheus.com.
5. The Kazaa file-sharing system. http://www.kazaa.com.
6. K. Aberer, P. Cudr´e-Mauroux, and M. Hauswirth. The chatty web: *Emergent semantics through gossiping*. In Proceedings of the 12th International World Wide Web Conference (WWW2003), Budapest, Hungary, May 2003.
7. P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. *Data management for peer-to-peer computing: A vision*. In Proceedings of the 5th International Workshop on the Web and Databases, Madison, Wisconsin, June 2002.
8. Y. Halevy, Z. G. Ives, P. Mork, and I. Tatarinov. *Piazza: Data management infrastructure for semantic web applications*. In Proceedings of the 12th International World Wide Web Conference (WWW2003), Budapest, Hungary, May 2003.
9. Alexander Löser, Wolfgang Nejdl, Martin Wolpers, and Wolf Siberski. *Information Integration in Schema-Based Peer-To-Peer Networks*. In Proceedings of the 15th International Conference of Advanced Information Systems Engieering (CAiSE 03), Klagenfurt, June 2003.
10. Arturo Crespo and Hector Garcia-Molina. *Semantic overlay networks, for P2P systems.* Technical Report, Stanford University, 2003.

11. C.-H. Ng, K.-C. Sia, and I. King. *Peer clustering and firework query model in the peer-to-peer network.* Technical Report, Chinese University of Hongkong, Department of Computer Science and Engineering, 2003.

12. Ingo Brunkhorst, Hadhami Dhraief, Alfons Kemper, Wolfgang Nejdl and Christian Wiesner. *Distributed Queries and Query Optimization in Schema-Based P2P Systems.* In International Workshop On Databases, Information Systems and Peer-to-Peer Computing, VLDB 2003, Berlin, Germany, September 2003.

13. Hadhami Dhraief, Alfons Kemper, Wolfgang Nejdl, and Christian Wiesner. *Processing and Optimization of Complex Queries in Schema-Based P2P Systems.* In Proceedings of the 2nd International Workshop On Databases, Information Systems and Peer-to-Peer Computing, Toronto, Canada, September 2004.

14. Ronald Fagin. *Combining fuzzy information from multiple systems.* In Proceedings of the 15th ACM Symposium on Principles of Database Systems, Montreal, 1996.

15. Hailing Yu, Huagang Li, Ping Wu, Divyakant Agrawal, Amr El Abbadi. *Efficient Processing of Distributed Top-k Queries.* In Proceedings of the 16th Database and Expert Systems Application (DEXA), 2005.

16. The JXTA Platform. http://www.jxta.org

17. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: *A scalable peer-to-peer lookup service for internet applications.* In Proceedings of the SIGCOMM 2001.

18. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. *A scalable content-addressable network.* In Proceedings of the SIGCOMM 2001.

19. Berners-Lee T., Hendler J., Lassila O. *The semantic web.* Scientific American, 284(5) pp. 34-43, May 2001.

20. Nejdl Wolfgang, Wolf Boris, Qu Changtao, Decker Stefan, Sintek Michael, Naeve Ambjorn, Nilsson Mikael, Palmer Matthias, Risch Tore. *EDUTELLA: A P2P Networking Infrastructure Based on RDF.* In Proceedings of the WWW2002, Honolulu, Hawaii, USA. May 7-11, 2002.

21. Grigoris Antoniou, Frank van Harmelen. ”*A Semantic Web Primer”*. MIT Press 2004.

22. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler (2000). E*xtensible Markup Language (XML) 1.0 (Second Edition)* W3C Recommendation, October 2000. Available at: http://www.w3.org/TR/2000/REC-xml-20001006/

23. O. Lassila, R Swick. *Resource Description Framework Model and Syntax Specification,* W3C Recommendation, Feb 1999, available at http://www.w3.org/TR/REC-rdf-syntax/

24. D. Brickley, R Guha. *Resource Description Framework Schema Specification 1.0,* W3C Candidate Recommendation, Mar 2000, available at http://www.w3.org/TR/rdf-schema/

25. M. Dean, G. Schreiber (eds), F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, L. Stein, *Web Ontology Language (OWL)*, available at http://www.w3.org/TR/owl-ref/

26. Wolfgang Nejdl, Martin Wolpers, Wolf Siberski, Alexander Löser, Ingo Bruckhorst, Mario Schlosser, and Christoph Schmitz. *Super-Peer-Based Routing and Clustering Strategies for RDF-Based Peer-To- Peer Networks.* In Proceedings of the 12th International World Wide Web Conference (WWW2003), Budapest, Hungary, May 2003.

27. Wolfgang Nejdl, Martin Wolpers, Wolf Siberski, Alexander Löser, Ingo Bruckhorst, Mario Schlosser, and Christoph Schmitz. S*uper-Peer-Based Routing Strategies for RDF-Based P2P Systems.* In Proceedings of the 2nd International Workshop On Databases, Information Systems and Peer-to-Peer Computing, Toronto, Canada, September 2004.

28. Wolfgang Nejdl, Mario Schlosser, Wolf Siberski, Martin Wolpers, Bernd, Simon, Stefan Decker, Michael Sintek. *RDF-based Peer-To-Peer-Networks for Distributed (Learning) Repositories.* Technical Report, November 2002

29. M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. *HyperCuP- Hypercubes, Ontologies and Efficient Search on P2P Networks.* In International Workshop on Agents and Peer-to-Peer Computing, Bologna, Italy, July 2002.

30. G. S. Manku and R. Motwani. *Approximate frequency counts over data streams.* In Proceedings of the 28th International Conference on Very Large Data Bases, Hong Kong, China, August 2002.

31. Alexander Löser, Martin Wolpers, Wolf Siberski, Wolfgang Nejdl. *Efficient data store discovery in a scientific P2P network.* In International Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data, In Proceedings of the ISWC 2003, Florida, USA, October 2003.

32. Alexander Löser, Felix Naumann, Wolf Siberski, Wolfgang Nejdl, and Uwe Thaden. *Semantic overlay clusters within super-peer networks.* In Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing in Conjunction with the VLDB 2003, Berlin, Germany, September 2003.

33. Christoph Tempich, Steffen Staab, and Adrian Wranik. *REMINDIN': Semantic query routing in peer-to-peer networks based on social metaphors.* In Proceedings of the 13th International WWW Conference, 2004.

34. David W. Aha, editor. *Lazy Learning.* Kluwer, Dordrecht, 1997.

35. Alexander Löser and Christoph Tempich. *On Ranking Peers in Semantic Overlay Networks.* In Proceedings of the 3rd Conference on Professional Knowledge Management, PAIKM'05, 2005.

36. Y. Li, Z.A. Bandar, and D. McLean. *An Approach for measuring semantic similarity between words using semantic multiple information sources.* In IEEE Transactions on Knowledge and Data Engineering, volume 15, 2003.

37. Steffen Staab, Alexander Löser and Christoph Tempich. *Semantic Methods for P2P Query Routing.* In Proceedings of the 3rd. German Conference on Multi Agent Technologies, LNAI 3550, Springer, 2005.

38. C. Tempich, A. Löser and J. Heinzmann. *Community based Ranking in Peer-to-Peer Networks.* In Proceedings of the 4th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE), Agia Napa, Cyprus 2005.

39. The SWAP Project. http://swap.semanticweb.org/

40. J. Broekstra. SeRQL: *Sesame RDF query language.* In M. Ehrig et al., editors, SWAP Deliverable 3.2 Method Design, pages 55–68. Available at: http://swap. semanticweb.org/public/Publications/swap-d3.2.pdf.

41. P. Haase, R. Siebes, F. Harmelen. *Peer selection in peer-to-peer networks with semantic topologies.* In Proceedings of the International Conference on Semantics of a Networked World: Semantics for Grid Databases, Paris, 2004.

42. George Kokkinidis and Vassilis Christophides. *Semantic Query Routing and Processing in P2P Database Systems: The ICS-FORTH SQPeer Middleware*. Third Hellenic Data Management Symposium (HDMS'04), Athens, Greece, June 28-29, 2004.

43. Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, Michel Scholl, *RQL: A Declarative Query Language for RDF*. In Proceedings of the 11th International World Wide Web Conference (WWW), Honolulu, Hawaii, USA, 2002.

44. Aimilia Magkanaraki, Val Tannen, Vassilis Christophides, Dimitris Plexousakis. *Viewing the Semantic Web through RVL Lenses.* In Proceedings of the 2nd International Semantic Web Conference (ISWC), 2003.

45. Vassilis Christophides, Gregory Karvounarakis, Ioanna Koffina, George Kokkinidis, Aimilia Magkanaraki, Dimitris Plexousakis, George Serfiotis, Val Tannen. *The ICS-FORTH SWIM: A Powerful Semantic Web Integration Middleware.* In Proceedings of the First International Workshop on Semantic Web and Databases (SWDB), Co-located with VLDB 2003, Humboldt-Universitat, Berlin, Germany, 2003.

46. A. Kemper and C. Wiesner. HyperQueries: *Dynamic Distributed Query Processing on the Internet.* In Proceedings of the Conference on Very Large Data Bases (VLDB), 2001.

47. Kemper, C. Wiesner, and P. Winklhofer. *Building dynamic market places using hyperqueries.* In Proceedings of the International Conference on Extending Database Technology, 2002.

48. V. Papadimos and D. Maier. *Distributed Query Processing and Catalogs for Peer-to-Peer Systems.* In Proceedings of the CIDR'03 International Conference, Asilomar, CA, USA, 2003.

49. R. Fagin et al.: O*ptimal aggregation algorithms for middleware.* In Symposium on Principles of Database Systems, 2001.

50. S. Chaudhuri and L. Gravano. *Evaluating top-k selection queries.* In Proceedings of the 25th International Conference on Very Large Data Bases (VLDB'99), 1999.

51. U. Güntzer, W.-T. Balke, W. Kießling. *Optimizing Multi-Feature Queries for Image Databases*. In Proceddings of the 26th International Conference on Very Large Databases (VLDB'00), 2000.

52. N. Bruno, S. Chaudhuri and L. Gravano. *Top-k selection queries over relational databases: Mapping strategies and performance evaluation*. In ACM Transactions on Database Systems, vol. 27, no. 2, June 2002.

53. Wolfgang Nejdl, Wolf Siberski, Uwe Thaden and Wolf-Tilo Balke. *Top-k Query Evaluation for Schema-Based Peer-to-Peer Networks.* In Proceedings of the 3rd International Semantic Web Conference (ISWC2004), Hiroshima, Japan, November 2004.

54. Ihab F. Ilyas, Walid G. Aref, Ahmed K. Elmagarmid: *Supporting top-k join queries in relational databases.* VLDB Journal 13(3): 207-221, 2004.

55. P. Tsaparas, T. Palpanas, Y. Kotidis, N. Koudas, and D. Srivastava. *Ranked join indices.* In Proceedings of ICDE, 2003.

56. Apostol Natsev, Yuan-Chi Chang, John R. Smith, Chung-Sheng Li, Jeffrey Scott Vitter: *Supporting Incremental Join Queries on Ranked Inputs.* In Proceedings of the VLDB 2001, pages 281-290, 2001.

57. M. Theobald, G. Weikum, and R. Schenkel. T*op-k query evaluation with probabilistic guarantees.* In Proceedings of VLDB 2004, pages 648–659, 2004.

58. S. Michel, P. Triantafillou, G.Weikum. Klee: *A framework for distributed top-k query algorithms.* In Proceedings of the VLDB 2005.

59. Katja Hose, Marcel Karnstedt, Kai-Uwe Sattler, Daniel Zinn. *Processing top-n queries in p2p-based web integration systems with probabilistic guarantees.* In Proceedings of WebDB 2005, Baltimore, Mariland, June 16-17, 2005.

60. Carlo Sartiani. *An Architecture for First-n and Top-n Queries in XML P2P Databases.* Universita di Pisa, 2005.

61. Witten, I., Moffat, A., Bell, T. *Managing Gigabytes*. Morgan Kaufman, Heidelberg ,1999.

62. Wolf-Tilo Balke, Wolfgang Nejdl, Wolf Siberski, and Uwe Thaden. *Progressive distributed top-k retrieval in peer-to-peer networks.* In Proceedings of ICDE'05, 2005.

63. N. Bruno, L. Gravano, A Marian. *Evaluating top-k queries over web-accessible databases.* In Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, IEEE Computer Society, 2002.

64. Amélie Marian, Nicolas Bruno, Luis Gravano. *Evaluating top-k queries over web-accessible databases.* ACM Transactions Database Systems. 29(2): 319-362, 2004.

65. P. Cao and Z. Wang. *Efficient Top-k Query Calculation in Distributed Networks.* In Proceedings of the 23rd annual ACM symposium on Principles of distributed computing, St. John's, Newfoundland, Canada, Pages 206-215, 2004.

66. B. Yang and H. Garcia-Molina. *Designing a super-peer network.* In Proceedings of the 19th International Conference on Data Engineering (ICDE'03).

67. Y. Tzitzikas, C. Meghini. *Query evaluation in peer-to-peer networks of taxonomy-based sources.* In Proceedings of the 10th International Conference on Cooperative Information Systems, CoopIS'03. Sicily, 2003.

68. Y. Tzitzikas, C. Meghini, N. Spyratos. *Taxonomy-based Conceptual Modeling for Peer-to-Peer Networks.* In Proceedings of the 22nd International Conference on Conceptual Modeling, ER'2003, Chicago, 2003.

69. Nikos Athanasis, Vassilis Christophides, Dimitris Kotzinos. Generating on the fly queries for the semantic web: The ICS-FORTH graphical RQL interface (GRQL). In Proceedings of the ISWC 2004.

70. Haase Peter, Broekstra Jeen, Eberhart Andreas, Volz Raphael: *A Comparison of RDF Query Languages.* S.A. McIlraith et al. (Eds.): In Proceedings of the ISWC 2004, LNCS 3298, pp. 502–517, 2004.

71. Comparison of RDF Query Languages. Available at: http://www.aifb.uni-karlsruhe.de/WBS/pha/rdf-query/

72. G. K. Zipf. *Human Behaviour and the Principle of Least Effort: an Introduction to Human Ecology*. Addison-Wesley, 1949.

73. Li Gong. *Project JXTA: A Technology Overview*. Sun Microsystems Inc, 2001.

74. Brendon J. Wilson. *"JXTA"*. New Riders Publishing, USA 2002.

75. *JXTA v.2.3.x. Java Programmer's Guide*. April 7, 2005. Available at: www.jxta.org/docs/JxtaProgGuide_v2.3.pdf

76. http://www.aim.com/

77. J. Nagel. *Congestion Control in TCP/IP Intemetworks*. Technical report, IETF Network Working Group, January 1984.

78. B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J.-C. Hugly, E. Pouyoul, B. Yeager. *Project JXTA 2.0 Super-Peer Virtual Network.* 2003. Available at: http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf

79. Gabriel Antoniu, Phil Hatcher, Mathieu Jan, and David A. Noblet. *Performance Evaluation of JXTA Communication Layers.* In Proceedings of the Workshop on Global and Peer-to-Peer Computing (GP2PC 2005), Cardiff, UK, May 2005.

80. The JXTA Distributed Framework project. http://jdf.jxta.org/.

81. http://jdf.jxta.org/juxtest/Efficiency/

82. http://java.sun.com/docs/books/tutorial/essential/threads/

83. http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Thread.html

84. The Eclipse Project http://www.eclipse.org

85. P. Garbacki, D. Epema, M. van Steen. *A Two-Level Semantic Caching Scheme for Super-Peer Networks.* In Proceedings of the IEEE 10th International Workshop on Web Content Caching and Distribution (WCW), Sophia Antipolis, France, September 2005.

86. Pinar Yolum, Munindar P. Singh. *Flexible Caching in Peer-to-Peer Information Systems.* In Proceedings of the CEUR Workshop, 2002.

87. http://wordnet.princeton.edu/

88. G.A. Miller, R Beckwith, C. Fellbaum, D. Gross, K. Miller. *Five papers on WordNet.* Technical Report CSL Report 43, Cognitive Systems Laboratory. Princeton University, 1990.