Query Processing in Super-Peer Networks with Languages Based on Information Retrieval: The P2P-DIET Approach*

Stratos Idreos[1], Christos Tryfonopoulos[1], Manolis Koubarakis[1], and Yannis Drougas[2]

[1] Intelligent Systems Laboratory, Dept. of Electronic and Computer Engineering, University of Crete, Greece
{sidraios,trifon,manolis}@intelligence.tuc.gr
[2] Dept. of Computer Science and Engineering, University of California Riverside, USA
drougas@cs.ucr.edu

Abstract. This paper presents P2P-DIET, an implemented resource sharing system that unifies one-time and continuous query processing in super-peer networks. P2P-DIET offers a simple data model for the description of network resources based on attributes with values of type text and a query language based on concepts from Information Retrieval. The focus of this paper is on the main modelling concepts of P2P-DIET (metadata, advertisements and queries), the routing algorithms (inspired by the publish/subscibe system SIENA) and the scalable indexing of resource metadata and queries.

1 Introduction

We consider the problem of *selective dissemination of information (SDI)* or *publish/subscribe* in peer-to-peer (P2P) networks [2, 7, 10, 14]. In an SDI scenario, a user posts a *continuous query* or *profile* to the system to receive notifications whenever certain *resources* of interest are *published*. Our work has culminated in the implementation of P2P-DIET [9, 11], a service that unifies one-time and continuous query processing in P2P networks with super-peers.

P2P-DIET combines one-time querying as found in other super-peer networks [16] and SDI as proposed in DIAS [10]. P2P-DIET has been implemented on top of the open source DIET Agents Platform[1] and it is currently available at `http://www.intelligence.tuc.gr/p2pdiet`. The new concept (in the data model) is that of advertisement. Other *new* features that distinguish P2P-DIET from DIAS and other recent systems [2, 7, 14] (client migration, dynamic IP addresses, stored notifications and rendezvous, fault-tolerance mechanisms, and message authentication and encryption) are not discussed and can be found in [8]. The contributions of this paper are the following:

* This work was supported in part by the European Commission projects DIET (5th Framework Programme IST/FET) and Evergrow (6th Framework Programme IST/FET). Christos Tryfonopoulos is partially supported by a Ph.D. fellowship from the program Heraclitus of the Greek Ministry of Education.

[1] `http://diet-agents.sourceforge.net/`

W. Lindner et al. (Eds.): EDBT 2004 Workshops, LNCS 3268, pp. 496–505, 2004.

- We briefly present our super-peer architecture, the protocols for handling advertisements, publications, queries, answers and notifications and discuss how they relate to the protocols of SIENA [2] and EDUTELLA [13].
- We introduce the *new filtering algorithm* BestFitTrie used by super-peers in P2P-DIET for matching textual resource metadata with continuous queries. We compare BestFitTrie with appropriate extensions of the algorithms used by SIFT [15], and discuss their relative strengths and weaknesses.

The rest of the paper is organized as follows. Section 2 briefly presents the metadata model and query language used for describing and querying resources in the current implementation of P2P-DIET. Section 3 discusses the protocols for processing advertisements, publications, queries, answers and notifications. Section 4 presents BestFitTrie and compares it with other alternatives. Section 5 concludes the paper. There is no related work section; instead, comparison of P2P-DIET with related systems is interspersed with our presentation.

2 The Data Model $\mathcal{AWP}$

In [12] we presented the data model $\mathcal{AWP}$ for specifying queries and *textual* resource metadata. Here, we give a brief description of the main concepts of $\mathcal{AWP}$ since it is the data model used in the rest of the paper. $\mathcal{AWP}$ is based on the concept of *attributes* with values of type *text*. The query language of $\mathcal{AWP}$ offers *Boolean* and *proximity operators* on attribute values as in the work of [3] which is based on the Boolean model of IR.

Let $\mathbf{\Sigma}$ be a finite *alphabet*. A *word* is a finite non-empty sequence of letters from $\mathbf{\Sigma}$. Let $\mathcal{A}$ be a countably infinite set of attributes called the *attribute universe*. In practice attributes will come from *namespaces* appropriate for the application at hand, e.g., from the set of Dublin Core Metadata Elements[2]. If $A \in \mathcal{A}$ then $\mathcal{V}_A$ denotes a set of words called the *vocabulary* of attribute A. A *text value* s of length n over a vocabulary $\mathcal{V}$ is a total function $s : \{1, 2, \ldots, n\} \rightarrow \mathcal{V}$.

A *publication* n is a set of attribute-value pairs (A, s) where $A \in \mathcal{A}$, s is a text value over $\mathcal{V}_A$, and all attributes are *distinct*. The following is a publication:

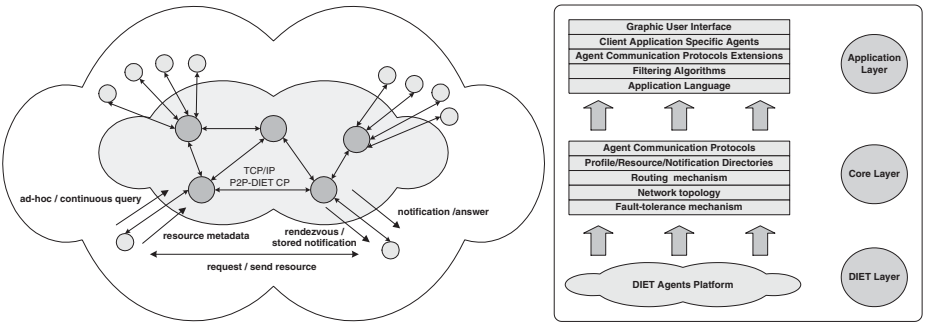
$$\{\ (AUTHOR, \text{“}John\ Smith\text{”}), (TITLE, \text{“}Information\ dissemination\ in\ P2P\ systems\text{”}),\ (ABSTRACT, \text{“}In\ this\ paper\ we\ show\ that\ ...\text{”})\ \}$$

A *query* is a conjunction of atomic formulas of the form $A = s$ or $A \sqsupseteq wp$, where wp is a word pattern containing conjunctions of words and proximity formulas with only words as subformulas, for example:

$$AUTHOR = \text{“}John\ Smith\text{”}\ \wedge TITLE \sqsupseteq p2p \wedge (information \prec_{[0,0]} dissemination)$$

The above query requests all resources that have *John Smith* as their author, and their title contains the word *p2p* and a word pattern where the word *information* is immediately followed by the word *dissemination*.

[2] `http://purl.org/dc/elements/1.1/`



(a) (b)

Fig. 1. The architecture and the layered view of P2P-DIET

Advertisements in P2P-DIET come in two kinds and are used to prune the paths of queries broadcasted in the super-peer network. An *attribute advertisement* is a subset of the attribute universe $\mathcal{A}$. An *attribute/value advertisement* is a set of pairs $(A, \{s_1, \ldots, s_n\})$ where every $A \in \mathcal{A}$ and every $s_1, \ldots, s_n$ are text values. Intuitively, advertisements are *intentional* descriptions of the content a peer expects to publish to the network. In this matter we follow SIENA [2] and EDUTELLA [13]. The former kind of advertisement gives only the attributes used by a peer to describe its content (e.g., a peer might use only $TITLE$ and $AUTHOR$), while the latter also lists the expected values of certain attributes (e.g., a peer might have only John Smith, John Brown and Tom Fox as authors). Attribute/value advertisements can be interpreted as disjunctions of equalities of the form $A = s_i$.

3 Architecture, Routing and Query Processing

A high-level view of the P2P-DIET architecture and its software layers is shown in Figure 1. There are two kinds of nodes: *super-peers* and *clients*. All super-peers are equal and have the same responsibilities, thus the super-peer subnetwork is a *pure* P2P network (it can be an arbitrary undirected graph). Each super-peer serves a fraction of the clients and keeps *indices* on the resources of those clients.

Resources (e.g., files in a file-sharing application) are kept at client nodes, although it is possible in special cases to store resources at super-peer nodes. Clients request resources directly from the resource owner client. A client is connected to the network through a single super-peer node, which is the *access point* of the client. It is not necessary for a client to be connected to the same access point continuously; *client migration* is supported in P2P-DIET. Clients can connect, disconnect or even leave from the system silently at any time. To enable a higher degree of decentralization and dynamicity, we also allow clients to use *dynamic IP addresses*. If clients are not on-line, notifications matching their interests are stored for them by their access points and delivered once

clients reconnect. If resource owners are not on-line, requesting clients can set up a rendezvous to obtain the required resources. Additionally, P2P-DIET supports a simple fault-tolerance protocol based on *are-you-alive* messages between super-peers, and between super-peers and their clients. Finally, P2P-DIET provides message authentication and message encryption using public key cryptography.

The super-peer subnetworks in P2P-DIET are expected to be more stable than typical pure P2P networks such as Gnutella. As a result, we have chosen to use routing algorithms appropriate for such networks, *shortest path trees* and *reverse path forwarding* [5].

Advertisements. P2P-DIET clients *advertise* the resources they expect to publish in the system by sending an advertisement message to their access point. Advertisements are then forwarded in the super-peer backbone. Whenever a resource is published by a client, P2P-DIET makes sure that it satisfies the advertisements made previously by the same client or else a new advertisement message, that contains the extra information for this client must be submitted to the client's access point and forwarded in the super-peer backbone.

One-Time Queries. In the typical *one-time query scenario*, a client *C* can pose a query *q* to its access point *AP* through a `query` message. The message contains the identifier of *C*, the IP address and port of *C* and the query *q*. *AP* broadcasts *q* to all super-peers using reverse path forwarding. The advertisements present at each super-peer are used to prune broadcasting paths. An attribute advertisement a blocks further propagation of query q if a does not cover q. An attribute/value advertisement a blocks further propagation of query q if q and a are inconsistent. *Answers* are produced for all matching network resources, by the super-peers that hold the appropriate resource metadata. A super-peer that generates an answer *a*, forwards *a* directly to *C* using the IP address and port of *C* included in the query *q*. Each super-peer can be understood to store a relation $resource(ID, A_1, A_2, \ldots, A_n)$ where ID is a resource identifier and $A_1, A_2, \ldots, A_n$ are the attributes of $\mathcal{A}$ used by the super-peer. In our implementation, relation $resource$ is implemented by keeping an *inverted file index* for each attribute A_i. The index maps every word w in the vocabulary of A_i to the set of resource IDs that contain word w in their attribute A_i. Query evaluation at each super-peer is then implemented efficiently by utilizing these indices in the standard way [1].

Continuous Queries. Clients may *subscribe* to their access point with a *continuous query*. Super-peers *forward* posed queries to other super-peers. Thus, matching a query with metadata of a published resource takes place at a super-peer that is *as close as possible* to the origin of the resource. A continuous query published by a client *C* is identified by the identifier of *C* and a very large random number, *query id* assigned by *C* at the time that the query was generated. A notification is generated at the access point AP_1 where the resource was published, and travels to the access point AP_2 of every client that has posted a continuous query matching this notification following the *shortest path* from AP_1 to AP_2. Then, the notification is delivered to the interested clients for further processing.

Each super-peer manages an *index* over its continuous queries. Using this index, a super-peer can generate notifications when resource metadata items are published by its

clients. Additionally, each super-peer manages a *continuous query poset* that keeps track of the subsumption relations among the continuous queries posted to the super-peer by its clients or forwarded by other super-peers. This poset is again inspired by SIENA [2] and it is used to minimize network traffic: in each super-peer no continuous query that is less general than one that has already been processed is actually forwarded.

4 Filtering Algorithms

In this section we present and evaluate BestFitTrie, a main memory algorithm that solves the filtering problem for *conjunctive queries* in $\mathcal{AWP}$. Because our work extends and improves previous algorithms of SIFT [15], we adopt terminology from SIFT in many cases.

BestFitTrie uses two data structures to represent each published document d: the *occurrence table* $OT(d)$ and the *distinct attribute list* $DAL(d)$. $OT(d)$ is a hash table that uses words as keys, and is used for storing all the attributes of the document in which a specific word appears, along with the positions that each word occupies in the attribute text. $DAL(d)$ is a linked list with one element for each distinct attribute of d. The element of $DAL(d)$ for attribute A points to another linked list, the *distinct word list* for A (denoted by $DWL(A)$) which contains all the distinct words that appear in $A(d)$.

To index queries BestFitTrie utilises an array, called the *attribute directory* (AD), that stores pointers to word directories. AD has one element for each distinct attribute in the query database. A *word directory* $WD(B_i)$ is a hash table that provides fast access to roots of *tries* in a *forest* that is used to organize *sets of words* – the set of words in wp_i (denoted by $words(wp_i)$) for each atomic formula $B_i \sqsupseteq wp_i$ in a query. The proximity formulas contained in each wp_i are stored in an array called the *proximity array* (PA). PA stores pointers to trie nodes (words) that are operands in proximity formulas along with the respective proximity intervals for each formula. Another hash table, called *equality table* (ET) indexes text values s_i that appear in atomic formulas of the form $A_i = s_i$.

When a new query q arrives, the index structures are populated as follows. For each attribute $A_i = s_i$, we hash text value s_i to obtain a slot in ET where we store the value A_i. For each attribute $B_j \sqsupseteq wp_j$, we compute $words(wp_j)$ and insert them in one of the tries with roots indexed by $WD(B_j)$. Finally, we visit PA and store pointers to trie nodes and proximity intervals for the proximity formulas contained in wp_j.

We now explain how each word directory $WD(B_j)$ and its forest of tries are organised. The idea is to store sets of words compactly by exploiting their *common elements*, to preserve memory space and to accelerate the filtering process.

Definition 1. *Let S be a set of sets of words and $s_1, s_2 \in S$ with $s_2 \subseteq s_1$. We say that s_2 is an* identifying subset *of s_1 with respect to S iff $s_2 = s_1$ or $\nexists\, r \in S$ such that $s_2 \subseteq r$.*

The sets of identifying subsets of two sets of words s_1 and s_2 with respect to a set S is the same if and only if s_1 is identical to s_2.

The sets of words $words(wp_j)$ are organised in the word directory $WD(B_j)$ as follows. Let S be the set of sets of words currently in $WD(B_j)$. When a new set of words s arrives, BestFitTrie selects an identifying subset t of s with respect to S and

uses it to organise s in $WD(B_j)$. The algorithm for choosing t depends on the current organization of the word directory and will be given below.

Throughout its existence, each trie T of $WD(B_j)$ has the following properties. The nodes of T store sets of words and other data items related to these sets. Let $sets\text{-}of\text{-}words(T)$ denote the set of all sets of words stored by the nodes of T. A node of T stores more than one set of words iff these sets are identical. The root of T (at depth 0) stores sets of words with an identifying subset of cardinality one. In general, a node n of T at depth i stores sets of words with an identifying subset of cardinality $i+1$. A node n of T at depth i storing sets of words equal to s is implemented as a structure consisting of the following fields:

- $Word(n)$: the $i+1$-th word w_i of identifying subset $\{w_0, \ldots, w_{i-1}, w_i\}$ of s where $w_0, \ldots, w_{i-1}$ are the words of nodes appearing earlier on the path from the root to node n.
- $Query(n)$: a linked list containing the identifier of query q that contained word pattern wp for which $\{w_0, \ldots, w_i\}$ is the identifying subset of $sets\text{-}of\text{-}words(T)$.
- $Remainder(n)$: if node n is a leaf, this field is a linked list containing the words of s that are not included in $\{w_0, \ldots, w_i\}$. If n is not a leaf, this field is empty.
- $Children(n)$: a linked list of pairs (w_{i+1}, ptr), where w_{i+1} is a word such that $\{w_0, \ldots, w_i, w_{i+1}\}$ is an identifying subset for the sets of words stored at a child of w_i and ptr is a pointer to the node containing the word w_{i+1}.

The sets of words stored at node n of T are equal to $\{w_0, \ldots, w_n\} \cup Remainder(n)$, where $w_0, \ldots, w_n$ are the words on the path from the root of T to n. An identifying subset of these sets of words is $\{w_0, \ldots, w_n\}$. The purpose of $Remainder(n)$ is to allow for the delayed creation of nodes in trie. This delayed creation lets us choose which word from $Remainder(n)$ will become the child of current node n depending on the sets of words that will arrive later on.

The algorithm for inserting a new set of words s in a word directory is as follows. The first set of words to arrive will create a trie with the first word as the root and the rest stored as the remainder. The second set of words will consider being stored at the existing trie or create a trie of its own. In general, to insert a new set of words s, BestFitTrie iterates through the words in s and utilises the hash table implementation of the word directory to find all *candidate tries* for storing s: the tries with root a word of s. To store sets as compactly as possible, BestFitTrie then looks for a trie node n such that the set of words $(\{w_0, \ldots, w_n\} \cup Remainder(n)) \cap s$, where $\{w_0, \ldots, w_n\}$ is the set of words on the path from the root to n, has maximum cardinality. There may be more than one node that satisfies this requirements and such nodes might belong to different tries. Thus BestFitTrie performs a depth-first search down to depth $|s| - 1$ in *all* candidate tries in order to decide the optimal node n. The path from the root to n is then extended with new nodes containing the words in $\tau = (s \setminus \{w_0, \ldots, w_n\}) \cap Remainder(n)$. If $s \subseteq \{w_0, \ldots, w_n\} \cup Remainder(n)$, then the last of these nodes l becomes a new leaf in the trie with $Query(l) = Query(n) \cup \{q\}$ (q is the new query from which s was extracted) and $Remainder(l) = Remainder(n) \setminus \tau$. Otherwise, the last of these nodes l points to two child nodes l_1 and l_2. Node l_1 will have $Word(l_1) = u$, where $u \in Remainder(n) \setminus \tau$, $Query(l_1) = Query(n)$ and $Remainder(l_1) = Remainder(n) \setminus (\tau \cup \{u\})$. Similarly node l_2 will have $Word(l_2) = v$, where $v \in$

$s \setminus (\{w_0, \ldots, w_n\} \cup \tau)$, $Query(l_2) = q$ and $Remainder(l_2) = s \setminus (\{w_0, \ldots, w_n\} \cup \tau \cup \{u\})$. The complexity of inserting a set of words in a word directory is *linear* in the size of the word directory but *exponential* in the size of the inserted set. This exponential dependency is not a problem in practice because we expect *queries to be small* and the crucial parameter to be the size of the query database.

The filtering procedure utilises two arrays named $Total$ and $Count$. $Total$ has one element for each query in the database and stores the number of atomic formulas contained in that query. Array $Count$ is used for counting how many of the atomic formulas of a query match the corresponding attributes of a document. Each element of array $Count$ is set to zero at the beginning of the filtering algorithm. If at algorithm termination, a query's entry in array $Total$ equals its entry in $Count$, then the query matches the published document, since all of its atomic formulas match the corresponding document attributes.

When a document d is published, BestFitTrie hashes the text value $C(d)$ contained in each document attribute C and probes the ET to find matching atomic formulas with equality. Then for each attribute C in $DAL(d)$ and for each word w in $DWL(C)$, the trie of $WD(C)$ with root w is traversed in a breadth-first manner. Only subtrees having as root a word contained in $C(d)$ are examined, and hash table $OT(d)$ is used to identify them quickly. At each node n of the trie, the list $Query(n)$ gives implicitly all atomic formulas $C \sqsupseteq wp$ that can potentially match $C(d)$ if the proximity formulas in wp are also satisfied. This is repeated for all the words in $DWL(C)$, to identify all the qualifying atomic formulas for attribute C. Then the proximity formulas for each qualifying query are examined using the polynomial time algorithm $prox$ from [12]. For each atomic formula satisfied by $C(d)$, the corresponding query element in array $Count$ is increased by one. At the end of the filtering algorithm the equal entries in arrays $Total$ and $Count$ give us the queries satisfied by d.

To evaluate the performance of BestFitTrie we have also implemented algorithms BF, SWIN and PrefixTrie. BF (Brute Force) has no indexing strategy and scans the query database sequentially to determine matching queries. SWIN (Single Word INdex) utilises a two-level index for accessing queries in an efficient way. PrefixTrie is an extension of the algorithm Tree of [15] appropriately modified to cope with attributes and proximity information. Tree was originally proposed for storing *conjunctions of keywords* in secondary storage in the context of the SDI system SIFT. Following Tree, PrefixTrie uses *sequences* of words sorted in lexicographic order for capturing the words appearing in the word patterns of atomic formulas (instead of sets used by BestFitTrie). A trie is then used to store sequences compactly by exploiting *common prefixes* [15].

Algorithm BestFitTrie constitutes an improvement over PrefixTrie. Because PrefixTrie examines only the prefixes of sequences of words in lexicographic order to identify common parts, it misses many opportunities for clustering. BestFitTrie keeps the main idea behind PrefixTrie but searches exhaustively the current word directory to discover the best place to introduce a new set of words. This allows BestFitTrie to achieve better clustering as PrefixTrie introduces redundant nodes that are the result of using a lexicographic order to identify common parts. This node redundancy can be the cause of deceleration of the filtering process as we will show in the next section. The only way to improve beyond BestFitTrie would be to consider *re-organizing* the word directory

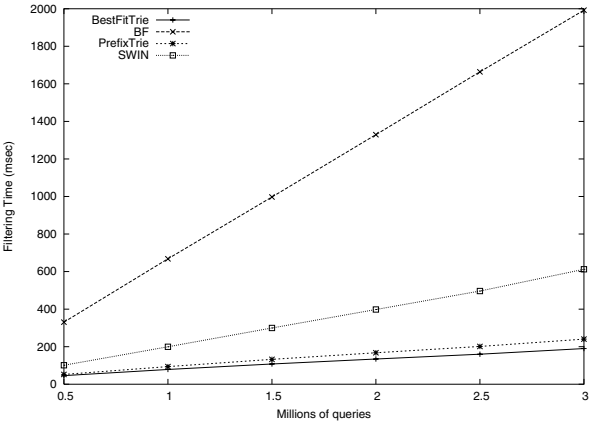


Fig. 2. Effect of the query database size in filtering time

every time a new set of words arrives, or periodically. We have not explored this approach in any depth.

4.1 Experimental Evaluation

We evaluated the algorithms presented above experimentally using a set of documents downloaded from ResearchIndex[3] and originally compiled in [6]. The documents are research papers in the area of Neural Networks and we will refer to them as the NN corpus. Because no database of queries was available to us, we developed a methodology for creating user queries using *words* and *technical terms* (phrases) extracted automatically from the ResearchIndex documents using the C-value/NC-value approach of [6].

All the algorithms were implemented in C/C++, and the experiments were run on a PC, with a Pentium III 1.7GHz processor, with 1GB RAM, running Linux. The results of each experiment are averaged over 10 runs to eliminate any fluctuations in the time measurements.

The first experiment targeted the performance of algorithms under different query database sizes. In this experiment, we randomly selected one hundred documents from the NN corpus and used them as incoming documents in query databases of different sizes. The size and the matching percentage for each document used was different but the average document size was 6869 words, whereas on average 1% of the queries stored matched the incoming documents.

As we can see in Figure 2, the time taken by each algorithm grows linearly with the size of the query database. However SWIN, PrefixTrie and BestFitTrie are less sensitive than Brute Force to changes in the query database size. The trie-based algorithms outperform SWIN mainly due to the clustering technique that allows the exclusion of more non-matching atomic queries. We can also observe that the better exploitation of the commonalities between queries improves the performance of BestFitTrie over PrefixTrie, resulting in a significant speedup in filtering time for *large query databases*.

[3] `http://www.researchindex.com`

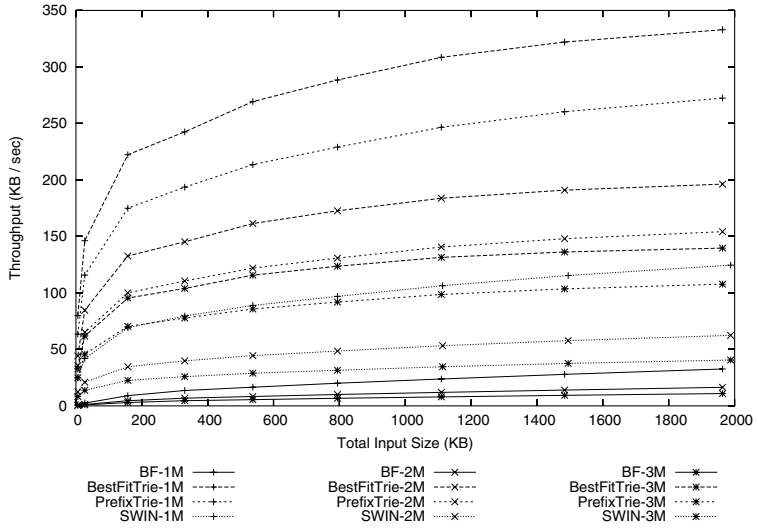


Fig. 3. Throughput for some algorithms for $\mathcal{AWP}$

Figure 3 shows that BestFitTrie gives the best filtering performance by processing a load of about 150KB (about 9 ResearchIndex papers) per second for a query database of 3 million queries.

In terms of space requirements BF needs about 15% less space than the trie-based algorithms, while the rate of increase for the two trie-based algorithms is similar to that of BF, requiring a fixed amount of extra space each time. Thus it is clear that BestFitTrie speeds up the filtering process with a small extra storage cost, and proves faster than the rest of the algorithms, managing to filter user queries about 10 times faster than the sequential scan method. Finally, the query insertion rate that the two trie-based algorithms can support is about 40 queries/second for a database containing 2.5 million queries.

We have also evaluated the performance of the algorithms under two other parameters: *document size* and *percentage of queries matching a published document*. Finally we have developed various heuristics for ordering words in the tries maintained by PrefixTrie and BestFitTrie when *word frequency* information (or *word ranking*) is available as it is common in IR research [1]. The details of these experiments are omitted due to space considerations.

5 Conclusions

We presented P2P-DIET, a service that unifies one-time and continuous query processing in P2P networks with super-peers. Currently our work concentrates on implementing the super-peer subnetwork of P2P-DIET using topologies with better properties and compare it analytically and experimentally with our current implementation. Our first steps in this direction are presented in [4].

References

1. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.
2. A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM TOCS*, 19(3):332–383, August 2001.
3. C.-C. K. Chang, H. Garcia-Molina, and A. Paepcke. Predicate Rewriting for Translating Boolean Queries in a Heterogeneous Information System. *ACM TOIS*, 17(1):1–39, 1999.
4. P.A. Chirita, S. Idreos, M. Koubarakis, and W. Nejdl. Publish/Subscribe for RDF-based P2P Networks. In *Proceedings of ESWS 2004*, May 2004.
5. Y.K. Dalal and R.M. Metcalfe. Reverse Path Forwarding of Broadcast Packets. *CACM*, 21(12):1040–1048, December 1978.
6. L. Dong. Automatic term extraction and similarity assessment in a domain specific document corpus. Master's thesis, Dept. of Computer Science, Dalhousie University, Halifax, Canada, 2002.
7. B. Gedik and L. Liu. PeerCQ:A Decentralized and Self-Configuring Peer-to-Peer Information Monitoring System. In *Proceedings of the 23rd ICDCS*, May 2003.
8. S. Idreos and M. Koubarakis. P2P-DIET: A Query and Notification Service Based on Mobile Agents for Rapid Implementation of P2P Applications. Technical report, TR-ISL-2003-01, Intelligent Systems Laboratory, Dept. of Electronic and Computer Engineering, Technical University of Crete, June 2003.
9. S. Idreos, M. Koubarakis, and C. Tryfonopoulos. P2P-DIET: Ad-hoc and Continuous Queries in Super-peer Networks. In *Proceedings of EDBT 2004*, volume 2992 of *LNCS*, pages 851–853, March 2004.
10. M. Koubarakis, T. Koutris, P. Raftopoulou, and C. Tryfonopoulos. Information Alert in Distributed Digital Libraries: The Models, Languages and Architecture of DIAS. In *Proceedings of ECDL 2002*, volume 2458 of *LNCS*, pages 527–542.
11. M. Koubarakis, C. Tryfonopoulos, S. Idreos, and Y. Drougas. Selective Information Dissemination in P2P Networks: Problems and Solutions. *ACM SIGMOD Record, Special issue on Peer-to-Peer Data Management, K. Aberer (editor)*, 32(3), September 2003.
12. M. Koubarakis, C. Tryfonopoulos, P. Raftopoulou, and T. Koutris. Data models and languages for agent-based textual information dissemination. In *Proceedings of CIA 2002*, volume 2446 of *LNCS*, pages 179–193, September 2002.
13. W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Loser. Super-peer based routing and clustering strategies for RDF-based peer-to-peer networks. In *Proceedings of the 12th WWW Conference*, 2003.
14. P.R. Pietzuch and J. Bacon. Peer-to-Peer Overlay Broker Networks in an Event-Based Middleware. In *Proceedings of DEBS'03*, June 2003.
15. T.W. Yan and H. Garcia-Molina. Index Structures for Selective Dissemination of Information Under the Boolean Model. *ACM TODS*, 19(2):332–364, 1994.
16. B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proceedings of ICDE 2003*, March 5–8 2003.