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, Department of Electronic and Computer Engineering, Technical University of Crete, 73100 Chania, Crete, Greece
{sidraios, trifon, manolis}@intelligence.tuc.gr
[2] Department of Computer Science and Engineering, University of California Riverside, CA 92521, USA
drougas@cs.ucr.edu

Abstract. This paper presents P2P-DIET, an implemented resource sharing system that unifies ad-hoc 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 form Information Retrieval. The focus of this paper is on the main modelling concepts of P2P-DIET (meta-data, advertisements and queries), the routing algorithms (inspired by the publish/subscibe system SIENA) and the scalable indexing of resource meta-data and queries.

1 Introduction

The main application scenario considered in recent peer-to-peer (P2P) data sharing systems is that of *ad-hoc querying*: a user poses a query (e.g., "I want music by Moby") and the system returns a list of pointers to matching files owned by various peers in the network. Then, the user can go ahead and download files of interest. The complementary scenario of *selective dissemination of information(SDI)* or *publish/subscribe* [6, 2] has started receiving attention only recently [2, 7, 16, 10, 15, 17]. 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* (e.g., when a song of Moby becomes available). SDI can be as useful as ad-hoc querying in many target applications of P2P networks ranging from file sharing, to more advanced applications such as alert systems for digital libraries, e-commerce networks etc.

At the Intelligent Systems Laboratory of the Technical University of Crete, we have recently concentrated on the problem of SDI in P2P networks in the

* This work was partially supported by project DIET (IST-1999-10088), within the UIE initiative of the IST/FET Programme of the European Commission.

context of project DIET[1]. Our work, summarized in [11], has culminated in the implementation of P2P-DIET, a service that unifies ad-hoc and continuous query processing in P2P networks with super-peers. P2P-DIET will be demonstrated at EDBT 2004 [9].

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.

Clients can run on user computers. 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 are equal to each other since the software running at each client node is equivalent in functionality. 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. P2P-DIET offers the ability to add or remove super-peers. Additionally, it supports a simple fault-tolerance protocol based on *are-you-alive* messages among super-peers, and among super-peers and their clients. Finally, P2P-DIET provides message authentication and message encryption using public key cryptography.

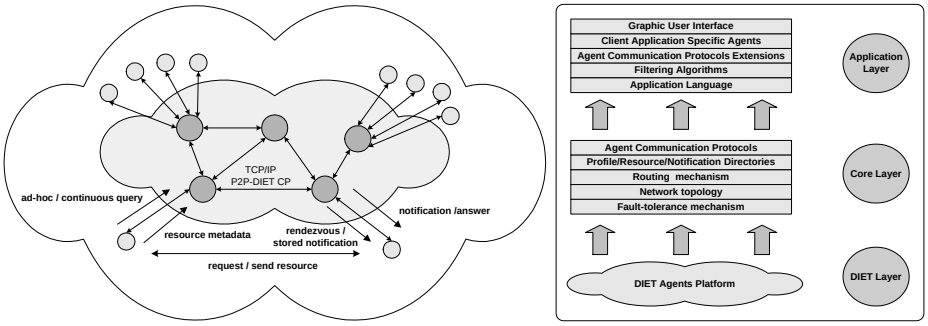
Conceptually, P2P-DIET is a direct descendant of DIAS, a Distributed Information Alert System for digital libraries, that was presented in [10] but was not fully implemented. P2P-DIET combines ad-hoc querying as found in other super-peer networks [19] and SDI as proposed in DIAS. P2P-DIET has been implemented on top of the open source DIET Agents Platform[2] and it is currently available at `http://www.intelligence.tuc.gr/p2pdiet`.

The contributions of this paper are the following:

- We review the P2P-DIET data model $\mathcal{AWP}$ for describing resources using *textual* meta-data. We show how to express formally *publications*, *queries* and *advertisements* in $\mathcal{AWP}$.
- We present briefly the super-peer architecture of P2P-DIET, the protocols for handling advertisements, publications, queries, answers and notifications and discuss how they relate to the protocols of SIENA [2] and EDUTELLA [14]. With respect to our earlier proposal DIAS [10], the new concept here (and in the data model) is that of advertisement. Other *new* features that

[1] `http://www.dfki.de/diet`

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



(a) (b)

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

distinguish P2P-DIET from DIAS and other recent systems [2, 7, 16, 15, 17] (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].

– We introduce the *new filtering algorithm* BestFitTrie used by super-peers in P2P-DIET for matching textual resource meta-data with continuous queries. We compare BestFitTrie with appropriate extensions of the algorithms used by the earlier SDI system SIFT [18], and discuss their relative strengths and weaknesses.

The rest of the paper is organized as follows. Section 2 presents the meta-data 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 [10] we have presented the data model $\mathcal{AWP}$ for specifying queries and *textual* resource meta-data in SDI systems such as P2P-DIET. $\mathcal{AWP}$ 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 Information Retrieval (IR).

Syntax. Let $\mathbf{\Sigma}$ be a finite *alphabet*. A *word* is a finite non-empty sequence of letters from $\mathbf{\Sigma}$. Let $\mathcal{V}$ be a (finite or countably infinite) set of words called the

vocabulary. A *text value* s of length n over vocabulary $\mathcal{V}$ is a total function $s : \{1, 2, \ldots, n\} \to \mathcal{V}$.

Let $\mathcal{I}$ be a set of *(distance) intervals* $\mathcal{I} = \{[l, u] : \ l, u \in \mathbb{N}, l \geq 0 \text{ and } l \leq u\} \cup \{[l, \infty) : \ l \in \mathbb{N} \text{ and } l \geq 0\}$. A *proximity formula* is an expression of the form $w_1 \prec_{i_1} \cdots \prec_{i_{n-1}} w_n$ where $w_1, \ldots, w_n$ are words of $\mathcal{V}$ and $i_1, \ldots, i_n$ are intervals of $\mathcal{I}$. Operators $\prec_i$ are called *proximity operators* and are generalizations of the traditional IR operators kW and kN [3]. Proximity operators are used to capture the concepts of *order* and *distance* between words in a text document. The proximity word pattern $w_1 \prec_{[l,u]} w_2$ stands for "word w_1 is *before* w_2 and is separated by w_2 by *at least* l and *at most* u *words*". The interpretation of proximity word patterns with more than one operator $\prec_i$ is similar. A *word pattern* over vocabulary $\mathcal{V}$ is a conjunction of words and proximity formulas. An example of a word pattern is $applications \wedge selective \prec_{[0,0]} dissemination \prec_{[0,3]} information$.

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[3].

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

$$\{ \ (AUTHOR, \text{"}John\ Smith\text{"}), \\ (TITLE, \text{"}Selective\ dissemination\ of\ information\ in\ P2P\ systems\text{"}), \\ (ABSTRACT, \text{"}In\ this\ paper\ we\ show\ that\ ...\text{"}) \ \}$$

A *query* is a conjunction of atomic formulas of the form $A = s$ or $A \sqsupseteq wp$ where $A \in \mathcal{A}$, s is a text value and wp is a word pattern. The following is a query:

$$AUTHOR = \text{"}John\ Smith\text{"} \wedge \\ TITLE \sqsupseteq (peer\text{-}to\text{-}peer \ \wedge (selective \prec_{[0,0]} dissemination \prec_{[0,3]} information))$$

Advertisements in P2P-DIET come in two kinds. 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 [14]. 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 $A = s_i$. As we will see in Section 3, advertisements are used to prune the paths of queries broadcasted in the super-peer network.

Semantics. The semantics of $\mathcal{AWP}$ have been defined in [10] and will not be presented here in detail. It is straightforward to define when a publication n

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

satisfies an atomic formula of the form $A = s$ or $A \sqsupseteq wp$, and then use this notion to define when n satisfies a query or an advertisement [10].

Let q_1 and q_2 be queries. We will say that q_2 *subsumes* q_1 if every publication n that satisfies q_1 also satisfies q_2. The notion of *advertisement subsumption* is defined in the same way.

Let a be an attribute advertisement and q be a query. We will say that a *covers* q if the set of attributes of q are a subset of a. The advertisement $a = \{AUTHOR, TITLE, ABSTRACT, BODY, DATE\}$ covers the example query given above.

Let a be an attribute/value advertisement and q be a query. We will say that a *covers* q if a subsumes q. We will say that a and q are *inconsistent* if there is no publication n that satisfies both. The advertisement $\{(AUTHOR, \{\text{“}John\ Smith\text{”}, \text{“}John\ Brown\text{”}, \text{“}Tom\ Fox\text{”}\})\}$ covers the example query given above. The advertisement $\{(AUTHOR, \{\text{“}John\ Brown\text{”}, \text{“}Tom\ Fox\text{”}\})\}$ is inconsistent with the above query.

Although $\mathcal{AWP}$ concentrates only on textual information, it is straightforward to extend it with numeric attributes (e.g., PRICE etc.).

3 Routing and Query Processing

P2P-DIET targets content sharing applications such as digital libraries [10] and networks of learning repositories [13]. Assuming that these applications are supported by P2P-DIET, we expect that there will be a stakeholder (e.g., a content provider such as Akamai) with an interest in building and maintaining the super-peer subnetwork. Thus 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. The rest of this section discusses these routing algorithms; the detailed protocols for super-peer join/leave, client join/leave, client migration, etc. are discussed in [8].

Advertisements. P2P-DIET clients advertise the resources they expect to publish in the system by sending an advertisement message to their access point. Advertisement forwarding in the super-peer backbone is then realized by the classic *reverse path forwarding* algorithm [20]: each advertisement message arriving at super-peer X through edge E is forwarded along all edges different than E if X believes that the best way to get to the source of the query is via E. This requires computation and storage of one-to-all shortest path information at each super-peer. Our measure of delay (weight on the edges of the network graph) is number of hops but we can easily implement other measures such as latency. As in SIENA, an *advertisement poset* is kept at each super-peer encoding advertisement subsumption as originally suggested in [2].

Ad-hoc queries. In the typical *ad-hoc query scenario*, a client C can post 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. SDI scenarios are also supported. Clients may *subscribe* to their access point with a *continuous query* expressing their information needs. Super-peers then *forward* posted queries to other super-peers. In this way, matching a query with meta-data 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.

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. Then, other clients with continuous queries matching this resource's meta-data are notified. 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 [8].

Each super-peer manages an *index* over its continuous queries. This index is used to solve the *filtering problem*: Given a database *db* of conjunctive continuous queries q expressed in the Boolean subset of $\mathcal{AWPS}$ and a resource meta-data item r, find all continuous queries $q \in db$ that match r. Using this index, a super-peer can generate notifications when resource meta-data 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. Using this data structure each super-peer in the shortest path that connects AP_1 to AP_2, finds out neighbouring super-peers or client-peers that have posted continuous queries that match n (less general than the query that fired the notification) and forwards n to them. The filtering

algorithm utilised in the current implementation of P2P-DIET is described in the following section.

We expect P2P-DIET networks to *scale* to very large numbers of clients, and high rates of advertisements, publications and queries. While a detailed performance analysis substantiating our claim is beyond the scope of this paper, we expect the indexing and poset data structures kept at each super-peer to be the main technical tools that help us to achieve scalability. To be able to scale to very large numbers of super-peers, we could substitute our arbitrary graph super-peer topology with one that guarantees efficient and non-redundant query broadcasts. In fact this is exactly what we have done in [4] using the hypercube topology HyperCup. Similar ideas have been presented in [17] where a distributed hash table topology based on Chord has been used.

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 for SIFT [18], 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. There is also a hash table, called *equality table* (ET), that indexes all text values s_i that appear in atomic formulas of the form $A_i = s_i$.

When a new query q of the form given above arrives, the index structures are populated as follows. For each attribute $A_i, 1 \leq i \leq n$, we hash text value s_i to obtain a slot in ET where we store the value A_i. For each attribute $B_j, 1 \leq j \leq m$, 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.

Id	Query $B_i \sqsupseteq wp_i$	Identifying Subsets
0	$B_i \sqsupseteq$ databases	{databases}
1	$B_i \sqsupseteq$ relational $\prec_{[0,2]}$ databases	{databases, relational}
2	$B_i \sqsupseteq$ databases $\wedge$ relational	{databases, relational}
3	$B_i \sqsupseteq$ (software $\prec_{[0,2]}$ neural $\prec_{[0,0]}$ networks) $\wedge$ (software $\prec_{[0,3]}$ relational $\prec_{[0,0]}$ databases)	{databases, relational, neural}, ...
4	$B_i \sqsupseteq$ optimal $\wedge$ (artificial $\prec_{[0,0]}$ intelligence) $\wedge$ relational $\wedge$ databases	{databases, relational, artificial, intelligence, optimal}, ...
5	$B_i \sqsupseteq$ artificial $\wedge$ relational $\wedge$ intelligence $\wedge$ databases $\wedge$ knowledge	{databases, relational, artificial, intelligence, knowledge }, ...

Table 1. Identifying subsets of $words(wp_i)$ with respect to $S = \{words(wp_i),\ i = 0, \ldots, 5\}$.

Let us now explain how each word directory $WD(B_j)$ and its forest of tries are organised. The main idea behind this data structure is to store sets of words compactly by exploiting their *common elements*. In this way, memory space is preserved and filtering becomes more efficient as we will see below.

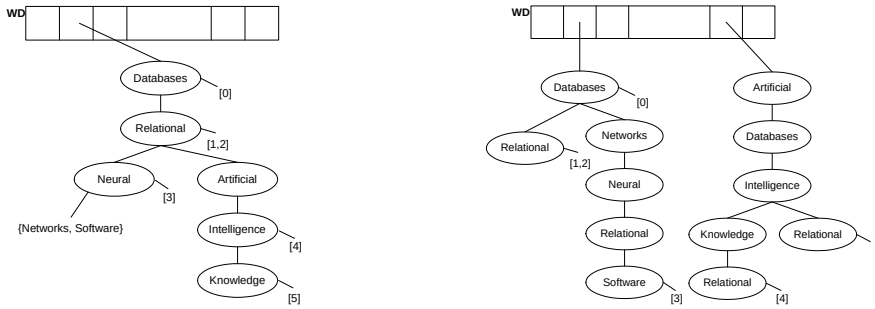
Definition 1. *Let S be a set of sets of words and $s_1, s_2 \in S$ with $s_2 \subseteq s_1$. We say that s_2 is an* identifying subset *of s_1 with respect to S iff $s_2 = s_1$ or $\nexists\ r \in S$ such that $s_2 \subseteq r$.*

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. Table 1 shows some examples that clarify these concepts.

The sets of words $words(wp_i)$ are organised in the word directory $WD(B_i)$ as follows. Let S be the set of sets of words currently in $WD(B_i)$. 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 $WD(B_i)$. 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 $WD(B_i)$ 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 if and only if 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 $sets\text{-}of\text{-}words(T)$.



(a) BestFitTrie (b) PrefixTrie

Fig. 2. BestFitTrie vs. PrefixTrie for the atomic queries of Table 1

- $Remainder(n)$: 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 $\{w_0, \ldots, w_n\}$. Figure 2(a) shows the general form of our index structure (we have omitted ET and PA). The part of $WD(B_i)$ corresponding to the queries of Table 1 is shown in full including lists $Query(n)$ and $Remainder(n)$. 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 (this is a standard data complexity assumption for an SDI environment).

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 at the server, filtering proceeds as follows. 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 $C \sqsupseteq wp_i$ that can potentially match $C(d)$ if the proximity formulas in wp_i 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 arrays $Total$ and $Count$ are traversed and the values stored for each query are compared. The equal entries in the two arrays 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. A query of the form presented at the beginning of this section is indexed by SWIN under all its attributes $A_1, \ldots, A_n, B_1, \ldots, B_m$ and also under n text values $s_1, \ldots, s_n$ and m words selected randomly from $wp_1, \ldots, wp_m$. More specifically SWIN utilises an ET to index equalities and an AD pointing

to several WDs to index the atomic containment queries. Atomic queries within a WD slot are stored in a list. PrefixTrie is an extension of the algorithm Tree of [18] 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* [18].

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 shown in Figure 2, where we can see that BestFitTrie needs only one trie to store the set of words for the formulas of Table 1, whereas 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 every time a new set of words arrives, or periodically. We have not explored this approach in any depth.

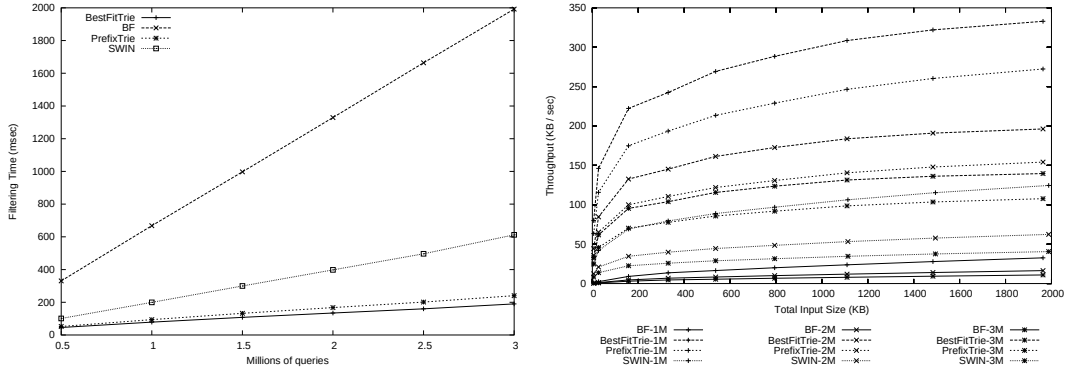
4.1 Experimental Evaluation

We evaluated the algorithms presented above experimentally using a set of documents downloaded from ResearchIndex[4] and originally compiled in [5]. 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 Research Index documents using the C-value/NC-value approach of [5].

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 time shown in the graphs is elapsed time in milliseconds and no other processes were run on the PC during the experiments.

The first experiment that we conducted to evaluate our algorithms targeted the performance of the four algorithms under different sizes of the query database. In this experiment we randomly selected one hundred documents from the NN corpus and used them as incoming documents in the query databases of different sizes. The size and the matching percentage for each document used

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



(a) (b)

Fig. 3. Effect of the query database size in filtering time and throughput

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 3(a), 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 filtering. 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*. Additionally, Figure 3(b) contrasts the algorithms in terms of throughput were we can see that BestFitTrie gives the best filtering performance managing to process 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, due to the simple data structure that poses small space requirements. Additionally 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. From the experiments above 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 as much as 3 million queries in less the 200 milliseconds, which is 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 ad-hoc 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 Transactions on Computer Systems*, 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 Transactions on Information Systems*, 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 the 1st European Semantic Web Symposium*, May 2004.
5. 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.
6. M.J. Franklin and S.B. Zdonik. "Data In Your Face": Push Technology in Perspective. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 516–519, 1998.
7. B. Gedik and L. Liu. PeerCQ:A Decentralized and Self-Configuring Peer-to-Peer Information Monitoring System. In *Proceedings of the the 23rd International Conference on Distributed Computing Systems*, 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 the IX International Conference on Extending Database Technology (EDBT04)*, March 2004.
10. M. Koubarakis, T. Koutris, C. Tryfonopoulos, and P. Raftopoulou. Information Alert in Distributed Digital Libraries: The Models, Languages and Architecture of DIAS. In *Proceedings of the 6th European Conference on Research and Advanced Technology for Digital Libraries (ECDL 2002)*, volume 2458 of *Lecture Notes in Computer Science*, pages 527–542, September 2002.

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 6th International Workshop on Cooperative Information Systems (CIA 2002)*, volume 2446 of *Lecture Notes in Computer Science*, pages 179–193, September 2002.
13. W. Nejdl, B. Wolf, Changtao Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch. Edutella: A P2P Networking Infrastructure Based on RDF. In *Proc. of WWW-2002*. ACM Press, 2002.
14. 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 International World Wide Web Conference*, 2003.
15. P.R. Pietzuch and J. Bacon. Peer-to-Peer Overlay Broker Networks in an Event-Based Middleware. In *Proceedings of the 2nd International Workshop on Distributed Event-Based Systems (DEBS'03)*, June 2003.
16. D. Tam, R. Azimi, and H.-Arno Jacobsen. Building Content-Based Publish/Subscribe Systems with Distributed Hash Tables. In *Proceedings of the 1st International Workshop On Databases, Information Systems and Peer-to-Peer Computing*, September 2003.
17. W.W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A.P. Buchmann. A Peer-to-Peer Approach to Content-Based Publish/Subscribe. In *Proceedings of the 2nd International Workshop on Distributed Event-Based Systems (DEBS'03)*, June 2003.
18. T.W. Yan and H. Garcia-Molina. Index structures for selective dissemination of information under the boolean model. *ACM Transactions on Database Systems*, 19(2):332–364, 1994.
19. B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proceedings of the 19th International Conference on Data Engineering (ICDE 2003)*, March 5–8 2003.
20. Y.K. Dalal and R.M. Metcalfe. Reverse path forwarding of broadcast packets. *Communications of the ACM*, 21(12):1040–1048, December 1978.