

# Publish/Subscribe Systems with Distributed Hash Tables and Languages from IR

## [Extended Abstract]\*

CHRISTOS TRYFONOPOULOS<sup>1</sup>  
*Technical University of Crete, Greece*

MANOLIS KOUBARAKIS  
*Technical University of Crete, Greece*

### Keywords

publish/subscribe, DHTs, indexing, performance.

## 1 Introduction

We consider the problem of implementing *publish/subscribe (pub/sub)* systems [3] on top of distributed hash tables (DHTs) such as Chord [11]. In pub/sub systems, clients post *continuous queries* (or *subscriptions*) to receive notifications whenever certain *resources* of interest are *published*. There has been lots of work in the area of pub/sub systems concentrating mostly on algorithms for the *filtering problem*: If  $Q$  is the set of queries posted to the network and  $p$  is a publication, how do we find efficiently which queries  $q \in Q$  match  $p$ ?

This work presents DH<sub>T</sub>rie, a distributed algorithm that uses a Chord-based protocol to distribute queries to super-peers and a *forest of tries* to index these queries at each super-peer. These data structures are used to solve the filtering problem efficiently. Our work adopts the super-peer architecture and the languages of P2P-DIET [8, 6]. Thus our pub/sub network has 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. Each super-peer serves a fraction of the clients. It is very easy to modify our proposal to work in the case of pure P2P networks where all nodes are equal. The fundamental difference of this work and our previous work on P2P-DIET is that DH<sub>T</sub>rie is based on ideas from distributed hash tables.

---

\*This work is partially supported by Integrated Project Evergrow (Contract No 001935) funded by the Complex Systems initiative of the FP6/IST/FET Programme of the European Commission.

<sup>1</sup>Christos Tryfonopoulos is partially supported by a Ph.D. fellowship from the program Heraclitus of the Greek Ministry of Education.

We use the data model  $\mathcal{AWP}$  of P2P-DIET for specifying queries and resource meta-data.  $\mathcal{AWP}$  is based on concepts from Information Retrieval (IR) : the concept of *attribute* with values of type *text* [9]. To the best of our knowledge, this is the first paper that discusses how to implement pub/sub systems with IR-based languages using ideas from DHTs. Other interesting work in this area which adopts different data models includes [10, 12].

The rest of this extended abstract is organised as follows. Section 3 discusses the distributed algorithm DHTrie. Section 4 briefly describes algorithm BestFitTrie that is the local component of DHTrie run by each super-peer and summarizes our experimental evaluation of BestFitTrie. Finally, Section 5 discusses work in progress.

## 2 The Data Model $\mathcal{AWP}$

In [9] we presented the data model  $\mathcal{AWP}$  for specifying queries and *textual* resource meta-data. We give here 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 [4] which is based on the Boolean model of IR.

Let  $\Sigma$  be a finite *alphabet*. A *word* is a finite non-empty sequence of letters from  $\Sigma$ . 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<sup>2</sup>. 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  $s : \{1, 2, \dots, n\} \rightarrow \mathcal{V}$ .

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:

$$\begin{aligned} & \{ (AUTHOR, \text{"John Smith"}), \\ & (TITLE, \text{"Information dissemination in P2P systems"}), \\ & (ABSTRACT, \text{"In this paper we show that ..."} ) \} \end{aligned}$$

A *query* is a conjunction of atomic formulas of the form  $A = s$  or  $A \sqsupseteq wp$ , where  $wp$  is a word pattern containing conjunction of words and proximity formulas with only words as subformulas. The following is an example of a query under  $\mathcal{AWP}$ :

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

The above query requests all resources that have *John Smith* as their author, and their title contains the word *peer-to-peer* and a word pattern where the word *selective* is immediately followed by the word *dissemination* which in turn is followed by the word *information* after at most three words.

<sup>2</sup><http://purl.org/dc/elements/1.1/>

### 3 The Algorithm DHTRie

DHTrie uses *three levels of indexing* to store continuous queries submitted by clients. The first level corresponds to the partitioning of the global query index to different super-peers using DHTs as the underlying infrastructure. Each super-peer is responsible for a fraction of the submitted user queries through a mapping of attribute-value combinations to super-peer identifiers. The distributed hash table infrastructure is used to define the mapping scheme and also manages the routing of messages between different super-peers.

The other two levels of our indexing mechanism are managed by each one of the super-peers, as they are used for indexing the user queries that a peer is responsible for. In the second level each super-peer uses a hash table to index the attributes contained in a query, whereas in the third level a trie-like structure that exploits commonalities between atomic queries is utilised.

#### 3.1 Mapping Keys to Super-Peers

We use a Chord-like DHT to implement our super-peer network. Chord [11] uses consistent hashing to map keys to nodes. Each node and data item is assigned an  $k$  bit identifier, where  $k$  is the length of an identifier that should be large enough to avoid the possibility of different items hashing to the same identifier. Identifiers can be thought of as being placed on a circle from 0 to  $2^k - 1$ , called the *identifier circle* or *Chord ring*. If  $H$  is the hash function used, then data item  $r$  is stored at the node with identifier  $H(r)$  if this node exists. Alternatively,  $r$  is stored at the node whose identifier is the first identifier *clockwise* in the Chord ring starting from  $H(r)$ . This node is called the *successor* of node  $H(r)$  and is denoted by  $successor(H(r))$ . We will say that this node is *responsible* for data item  $r$ .

#### 3.2 Subscribing With a Continuous Query

Let us assume that a client  $C$  wants to submit a continuous query  $q$  of the form:

$$A_1 = s_1 \wedge \dots \wedge A_m = s_m \wedge \\ A_{m+1} \supseteq wp_{m+1} \wedge \dots \wedge A_n \supseteq wp_n$$

$C$  contacts a super-peer  $S$  (its *access point*) and sends it a `SUBMITCQUERY( $id(C)$ ,  $q$ )` message, where  $id(C)$  is a unique identifier assigned to  $C$  by  $S$  in their first communication. When  $S$  receives  $q$ , it selects a random attribute  $A_i$ ,  $1 \leq i \leq n$  contained in  $q$  and a random word  $w_j$  from text value  $s_i$  or word pattern  $wp_i$  (depending on what kind of atomic formula of query  $q$  attribute  $A_i$  appears in). Then  $S$  forms the concatenation  $A_i w_j$  of strings  $A_i$  and  $w_j$  and computes  $H(A_i w_j)$  to obtain a super-peer identifier. Finally,  $S$  creates `FWDCQUERY( $id(S)$ ,  $id(q)$ ,  $q$ )` message and forwards it to super-peer with identifier  $H(A_i w_j)$  using the routing infrastructure of the DHT.

When a super-peer receives a FWDCQUERY message containing  $q$ , it inserts  $q$  in its local data structures using the insertion algorithm of BestFitTrie described briefly in Section 4 and also in [13, 7].

### 3.3 Publishing a Resource

When client  $C$  wants to publish a resource, it constructs a publication  $p$  of the form  $\{(A_1, s_1), (A_2, s_2), \dots, (A_n, s_n)\}$ , it contacts a super-peer  $S$  and sends  $S$  a PUBLISH(id( $C$ ),  $p$ ) message. When  $S$  receives  $p$ , it computes a list of super-peer identifiers that are provably a superset of the set of super-peer identifiers responsible for queries that match  $p$ . This list is computed as follows. For every attribute  $A_i$ ,  $1 \leq i \leq n$  in  $p$ , and every word  $w_j$  in  $s_i$ ,  $S$  computes  $H(A_i w_j)$  to obtain a list of super-peer identifiers that, according to the DHT mapping function, store continuous queries containing word  $w_j$  in the respective text value  $s_i$  or word pattern  $wp_i$  of attribute  $A_i$ .  $S$  then sorts this list in ascending order starting from  $id(S)$  to obtain list  $L$  and creates a FWDRESOURCE(id( $S$ ), id( $p$ ),  $p$ ,  $L$ ) message, where  $id(p)$  is a unique metadata identifier assigned to  $p$  by  $S$ , and sends it to super-peer with identifier equal to  $head(L)$ . This forwarding is done as follows: message FWDRESOURCE is sent to a super-peer  $S'$ , where  $id(S')$  is the greatest identifier contained in the finger table of  $S$ , for which  $id(S') \leq head(L)$  holds.

Upon reception of a FWDRESOURCE message by a super-peer  $S$ ,  $head(L)$  is checked. If  $id(S) = head(L)$  then  $S$  removes  $head(L)$  from list  $L$  and makes a copy of the message. The publication part of this message is then matched with the super-peer's local query database and subscribers are notified (the details of this are presented in Section 3.4). Finally,  $S$  forwards the message to super peer with identifier  $head(L)$ . If  $id(S)$  is not in  $L$ , then it just forwards the message as described in the previous paragraph.

### 3.4 Notifying Interested Subscribers

Let us now examine how notifications about published resources are sent to interested subscribers. When a FWDRESOURCE message containing a publication  $p$  of a resource arrives at a super-peer  $S$ , the continuous queries matching  $p$  are found by utilising its local index structures and using the algorithm BestFitTrie briefly described in Section 4 and also in [13, 7].

Once all the matching queries have been retrieved from the database,  $S$  creates a notification message of the form CQNOTIFICATION(id( $C$ ),  $l(r)$ ,  $L$ ,  $T$ ), where  $l(r)$  is a link to the resource,  $L$  is a list of identifiers of the super-peers that are intended recipients of the notification message, and  $T$  is a list containing the query identifiers of the queries that matched  $p$ . List  $L$  is created as follows.  $S$  finds all super-peers that have at least one client with a query  $q$  satisfied by  $p$ . Then it sorts the list in ascending order starting from  $id(S)$  and removes duplicate entries. The notification message is then forwarded according to the algorithm described in

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

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

### Section 3.3.

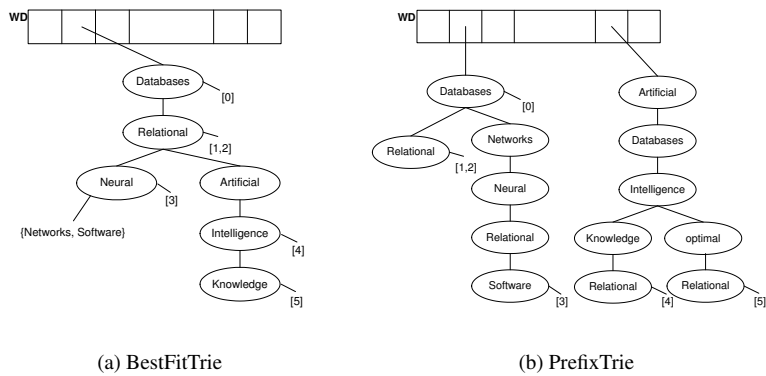
Upon arrival of a CQNOTIFICATION message at a super-peer  $S$ ,  $head(L)$  is checked to find out whether  $S$  is an intended recipient of the message. If it is not,  $S$  just forwards the message to another super-peer using information from its finger table and the algorithm described in Section 3.3. If  $head(L) = id(S)$ , then  $S$  scans  $T$  to find the set  $U$  of query identifiers that belong to clients that have  $S$  as their access point, by utilising a hash table that associates query identifiers with client identifiers. For each distinct query identifier in set  $U$ , a MATCHINGRESOURCE( $id(S), id(q), l(r)$ ) message is created and forwarded to the appropriate client. Finally  $S$  removes  $head(L)$  from  $L$  and  $U$  from  $T$ , and forwards CQNOTIFICATION message according to the algorithm described in Section 3.3.

## 4 Local Algorithms and Data Structures

In this section we describe the indexing structures that are used locally by each super-peer to store the continuous queries it is responsible for. These indexing structures are utilised in the step of DHtrie described in Section 3.4 to find efficiently which queries match a given publication.

### 4.1 The Algorithm BestFitTrie

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. For a query of the form presented in Section 3.2, a



(a) BestFitTrie

(b) PrefixTrie

Figure 1: Organisation of the atomic queries of Table 1

word directory  $WD(A_i)$ ,  $m + 1 \leq i \leq n$  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  $wp_i$  for each atomic formula  $A_i \sqsubseteq wp_i$  in a query (denoted by  $words(wp_i)$ ). 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  $s_j$ ,  $1 \leq j \leq m$  that appear in atomic formulas of the form  $A_j = s_j$ . The main idea behind BestFitTrie is to identify *common parts* between queries (through the use of identifying subsets; an example is shown in Table 1) and to exploit these commonalities to achieve faster filtering. BestFitTrie is described in detail in [13].

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 Section 3.2, is indexed by SWIN under all its attributes  $A_1, \dots, A_n$  and also under  $m$  text values  $s_1, \dots, s_m$  and  $n - m$  words selected randomly from  $wp_m + 1, \dots, wp_n$ . More specifically SWIN utilises an *ET* to index equalities and an *AD* pointing to several *WD*s 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 [14] 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* [14].

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 (see Figure 1(b) where the trie constructed by PrefixTrie for the example queries of Table 1 is shown). BestFitTrie keeps the main idea behind PrefixTrie but (i) handles the words contained in a query as a set rather than as a sorted sequence and (ii) searches exhaustively the forest of tries to discover the best place to introduce a new set of words. This allows BestFitTrie to achieve better clustering as shown in Figure 1, where we can see that it 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. To improve beyond BestFitTrie it would be interesting to consider *re-organizing* the word directory every time a new set of words arrives, or periodically, but this might turn out to be prohibitively expensive. In this work we have not explored this approach in any depth.

## 4.2 Experimental Evaluation

To evaluate the performance of our local indexing structures we used set of documents downloaded from ResearchIndex<sup>3</sup> 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 first experiment that we conducted to evaluate BestFitTrie targeted the performance 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 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

---

<sup>3</sup><http://www.researchindex.com>

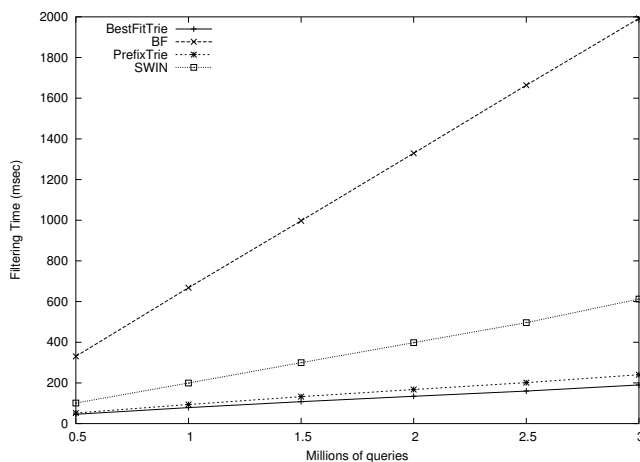


Figure 2: Effect of the query database size in filtering time

trie-based algorithms outperform SWIN mainly due to the clustering technique that allows the exclusion of more non-matching atomic queries during 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* (BestFitTrie is 20% faster than PrefixTrie and 1000% faster than sequential scan for a database of 3 million queries). Additionally, Figure 3 contrasts the algorithms in terms of query insertion time. In this figure we can see the average time in milliseconds needed to insert a new query in a query database of different sizes. Notice that in the case of 2.5 millions of queries BestFitTrie needs 5 milliseconds more to insert a query into the database, to save about 45 milliseconds at filtering time.

Comparison of the algorithms in terms of throughput results in BestFitTrie giving 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 than 200 milliseconds, which is about 10 times faster than the sequential scan method.

We have also evaluated the performance of the algorithms under two other parameters: *document size* and *percentage of queries matching a published docu-*



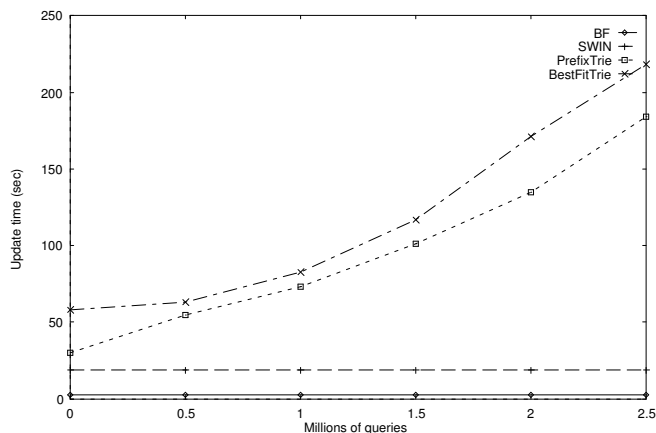


Figure 3: Query insertion time for different query database sizes

ment. Document size does not appear to significantly affect trie-based algorithms mainly due to the data structures used for the representation of an incoming document and the way the matching process is carried out. On the other hand the percentage of the stored queries that match an incoming document seem to have less effect on SWIN. In this experiment BestFitTrie seemed more sensitive than PrefixTrie to the matching percentage, but still proved more efficient in terms of filtering time and throughput.

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 [2]. Using the *inverse rank* heuristic (*irank*) [14], we store the least frequent words of the queries near the roots of the tries, while the frequent ones are pushed deeper in the tries, resulting in many narrow tries. Thus more queries are put in subtrees of words occurring less frequently, resulting in less lookups during filtering time. The algorithms using the *irank* heuristic are PrefixTrie-*irank* and BestFitTrie-*irank*. The faster algorithm is shown to be a variation of BestFitTrie, called LCWTrie (Least Common Word), where BestFitTrie is limited to consider a single candidate trie during query insertion: the one that has the least frequent word of the atomic query as root. The details of the experiments briefly mentioned above are presented in detail in [13].

## 5 Work in Progress

**Performance evaluation in the distributed case.** To evaluate the performance and scalability of DHtrie we are currently implementing the algorithms of Sec-

tion 3. We plan to evaluate DHTrie by considering its behaviour (mainly expressed in terms of message load between super-peers) under various parameters (query and resource size, arrival rates of queries and resources, number of super-peers, etc.). Furthermore we plan to extend the algorithm to take locality issues into account by maintaining extra routing tables for the most frequently contacted super-peers (namely the super-peers responsible for the most frequent words published).

**Load balancing.** A key problem that arises when trying to partition the query space among the different super-peers in our overlay network is *load balancing*. The idea here is to avoid having overloaded peers i.e., peers having to handle a great number of posted queries (this is what [1] calls *storage load balancing*; although the paper [1] is not in a pub/sub setting, the concept is the same).

In addition, we would like to have a way to deal with the load balancing problem posed to super-peers that are responsible for pairs  $(A, w)$ , where word  $w$  appears frequently in text values involving  $A$ . We expect the frequency of occurrence of words appearing in a query within an atomic formula with attribute  $A$  to follow a non-uniform distribution (e.g., a skewed distribution like the Zipf distribution [15]). We do not know of any study that has shown this by examining collections of user queries; however, such an assumption seems intuitive especially in the light of similar distributions of words in text collections [2]. As an example, in a digital library application we would expect distinguished author names to appear frequently in queries with the AUTHOR attribute, or popular topics to appear frequently in queries with the TITLE attribute. Thus in our case, uniformity of data items (i.e., queries) as traditionally assumed by DHTs is not applicable.

We are currently working on addressing the above load balancing problems by utilizing ideas from the algorithm LCWTrie described in detail in [13, 7] where queries are indexed under infrequent words; we also use a form of controlled replication to deal with overloading due to notification processing. It would be interesting to compare this approach to what is advocated in [1].

**Word frequency computation in a distributed setting.** Computing the frequency of occurrence of words in a distributed setting is a crucial problem if one wants to support vector space queries or to provide for load balancing among peers as showed above. There are mainly two approaches to the word frequency computation in a distributed setting; (a) a global ranking scheme that assumes a central authority that maintains the frequency information or a message-intensive update mechanism that notifies every peer in the network about changes in frequency information or (b) a local ranking scheme that computes word frequencies of peer  $p_i$  based solely on frequencies of words in documents that are published at  $p_i$ . It is clear that the first approach affects the scalability and efficiency of the system while the second approach can be misleading due to peer specialisation.

In related work we present a distributed word ranking algorithm that is a hybrid form of the two approaches described earlier. It provides an algorithm that is based on local information, but also tries to combine this information with the

global “truth” through an updating and estimation mechanism.

**Reducing network traffic.** We can reduce network traffic by *compressing* publications. In the full version of the paper we describe a gap compression technique that allows the matching of a compressed publication against a database of user queries using algorithm BestFitTrie.

**Christos Tryfonopoulos** is with the Dept. of Electronic and Computer Engineering, Technical University of Crete, 73100 Chania, Crete, Greece. E-mail: trifon@intelligence.tuc.gr

**Manolis Koubarakis** is with the Dept. of Electronic and Computer Engineering, Technical University of Crete, 73100 Chania, Crete, Greece. E-mail: manolis@intelligence.tuc.gr

## References

- [1] ABERER, K., DATTA, A., AND HAUSWIRTH, M. Multifaceted Simultaneous Load Balancing in DHT-based P2P systems: A new game with old balls and bins. Tech. Rep. IC/2004/23, Swiss Federal Institute of Technology Lausanne (EPFL), 2004.
- [2] BAEZA-YATES, R., AND RIBEIRO-NETO, B. *Modern Information Retrieval*. Addison Wesley, 1999.
- [3] CARZANIGA, A., ROSENBLUM, D.-S., AND WOLF, A. Design and evaluation of a wide-area event notification service. *ACM TOCS* 19, 3 (August 2001), 332–383.
- [4] CHANG, C.-C. K., GARCIA-MOLINA, H., AND PAEPCKE, A. Predicate Rewriting for Translating Boolean Queries in a Heterogeneous Information System. *ACM TOIS* 17, 1 (1999), 1–39.
- [5] DONG, L. 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] IDREOS, S., KOUBARAKIS, M., AND TRYFONOPOULOS, C. P2P-DIET: Ad-hoc and Continuous Queries in Super-peer Networks. In *Proceedings of EDBT 2004* (Heraklion, Crete, Greece, 14–18 March 2004).
- [7] IDREOS, S., TRYFONOPOULOS, C., KOUBARAKIS, M., AND DROUGAS, Y. Query Processing in Super-Peer Networks with Languages Based on Information Retrieval: the P2P-DIET Approach. In *Proceedings of P2P & DB 2004* (2004).
- [8] KOUBARAKIS, M., TRYFONOPOULOS, C., IDREOS, S., AND DROUGAS, Y. 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).

- [9] KOUBARAKIS, M., TRYFONOPOULOS, C., RAFTOPOULOU, P., AND KOUTRIS, T. Data models and languages for agent-based textual information dissemination. In *Proceedings of CIA 2002* (September 2002), vol. 2446 of *LNCS*, pp. 179–193.
- [10] PIETZUCH, P., AND BACON, J. Peer-to-Peer Overlay Broker Networks in an Event-Based Middleware. In *Proceedings of DEBS'03* (June 2003).
- [11] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M., AND BALAKRISHNAN, H. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the ACM SIGCOMM '01 Conference* (San Diego, California, August 2001).
- [12] TERPSTRA, W., BEHNEL, S., FIEGE, L., ZEIDLER, A., AND BUCHMANN, A. A Peer-to-Peer Approach to Content-Based Publish/Subscribe. In *Proceedings of DEBS'03* (June 2003).
- [13] TRYFONOPOULOS, C., KOUBARAKIS, M., AND DROUGAS, Y. Filtering Algorithms for Information Retrieval Models with Named Attributes and Proximity Operators. Submitted to a conference. Under review, 2004.
- [14] T.W. YAN AND H. GARCIA-MOLINA. Index structures for selective dissemination of information under the boolean model. *ACM TODS* 19, 2 (1994), 332–364.
- [15] ZIPF, G. K. *Human Behaviour and Principle of Least Effort*. Addison Wesley, Cambridge, Massachusetts, 1949.