Publish/Subscribe Systems with Distrib
Hash Tables and Languages from IF
[Extended Abstract]*

CHRISTOS TRYFONOPOULOS[1]
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)*
[3] on top of distributed hash tables (DHTs) such as Chord [11]. In pub/
tems, clients post *continuous queries* (or *subscriptions*) to receive noti
whenever certain *resources* of interest are *published*. There has been lots
in the area of pub/sub systems concentrating mostly on algorithms for the
problem: If Q is the set of queries posted to the network and p is a pub
how do we find efficiently which queries $q \in Q$ match p?

This work presents DHTrie, a distributed algorithm that uses a Cho
protocol to distribute queries to super-peers and a *forest of tries* to ind
queries at each super-peer. These data structures are used to solve the
problem efficiently. Our work adopts the super-peer architecture and
guages of P2P-DIET [8, 6]. Thus our pub/sub network has two kinds o
super-peers and *clients*. All super-peers are equal and have the same r
bilities, thus the super-peer subnetwork is a *pure* P2P network. Each su
serves a fraction of the clients. It is very easy to modify our proposal to
the case of pure P2P networks where all nodes are equal. The fundament
ence of this work and our previous work on P2P-DIET is that DHTrie is
ideas from distributed hash tables.

*This work is partially supported by Integrated Project Evergrow (Contract No 001935)
the Complex Systems initiative of the FP6/IST/FET Programme of the European Commiss
[1]Christos Tryfonopoulos is partially supported by a Ph.D. fellowship from the program
of the Greek Ministry of Education.

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

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

2 The Data Model $\mathcal{AWP}$

In [9] we presented the data model $\mathcal{AWP}$ for specifying queries and *te* source meta-data. We give here a brief description of the main concepts c since it is the data model used in the rest of the paper. $\mathcal{AWP}$ is base concept of *attributes* with values of type *text*. The query language of $\mathcal{AW}$ *Boolean* and *proximity operators* on attribute values as in the work of [4 is based on the Boolean model of IR.

Let Σ be a finite *alphabet*. A *word* is a finite non-empty sequence c from Σ. Let $\mathcal{A}$ be a countably infinite set of attributes called the *attribute u* In practice attributes will come from *namespaces* appropriate for the ap at hand e.g., from the set of Dublin Core Metadata Elements[2]. If $A \in \mathcal{A}$ denotes a set of words called the *vocabulary* of attribute A. A *text value s* c 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 value over $\mathcal{V}_A$, and all attributes are *distinct*. The following is a publicati

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

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

$$\begin{array}{c}(AUTHOR = \text{“}John\ Smith\text{”}) \wedge\\ (TITLE \sqsupseteq (peer\text{-}to\text{-}peer\ \wedge (selective \prec_{[0,0]} dissemination \prec_{[0,3]} informati\end{array}$$

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

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

3 The Algorithm DHTrie

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

The other two levels of our indexing mechanism are managed by e
of the super-peers, as they are used for indexing the user queries that
responsible for. In the second level each super-peer uses a hash table to i
attributes contained in a query, whereas in the third level a trie-like struc
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 [
consistent hashing to map keys to nodes. Each node and data item is ass
k bit identifier, where k is the length of an identifier that should be large e
avoid the possibility of different items hashing to the same identifier. Id
can be thought of as being placed on a circle from 0 to $2^k - 1$, called the
circle or *Chord ring*. If H is the hash function used, then data item r is
the node with identifier $H(r)$ if this node exists. Alternatively, r is store
node whose identifier is the first identifier *clockwise* in the Chord ring
from $H(r)$. This node is called the *successor* of node $H(r)$ and is de
$successor(H(r))$. We will say that this node is *responsible* for data item

3.2 Subscribing With a Continuous Query

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

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

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

When a super-peer receives a FWDCQUERY message containing q, q in its local data structures using the insertion algorithm of BestFitTrie d 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 $\{(A_1,s_1),(A_2,s_2),\ldots,(A_n,s_n)\}$, it contacts a super-peer S and sends S a SOURCE$(id(C),p)$ message. When S receives p, it computes a list of su identifiers that are provably a superset of the set of super-peer identifiers sible for queries that match p. This list is computed as follows. For every 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 obt of super-peer identifiers that, according to the DHT mapping function, st tinuous queries containing word w_j in the respective text value s_i or word wp_i of attribute A_i. S then sorts this list in ascending order starting fro to obtain list L and creates a FWDRESOURCE$(id(S),id(p),p,L)$ messag $id(p)$ is a unique metadata identifier assigned to p by S, and sends it to su with identifier equal to $head(L)$. This forwarding is done as follows: FWDRESOURCE is sent to a super-peer S', where $id(S')$ is the greatest i 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, *he* checked. If $id(S) = head(L)$ then S removes $head(L)$ from list L and copy of the message. The publication part of this message is then match the super-peer's local query database and subscribers are notified (the d this are presented in Section 3.4). Finally, S forwards the message to su with identifier $head(L)$. If $id(S)$ is not in L, then it just forwards the me described in the previous paragraph.

3.4 Notifying Interested Subscribers

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

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

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

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

Section 3.3.

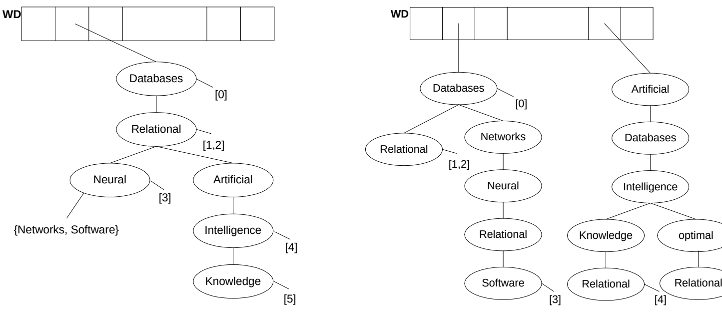
Upon arrival of a CQNOTIFICATION message at a super-peer S, *he*
checked to find out whether S is an intended recipient of the message. If
S just forwards the message to another super-peer using information fro
ger table and the algorithm described in Section 3.3. If $head(L) = id(S$
scans T to find the set U of query identifiers that belong to clients that h
their access point, by utilising a hash table that associates query identif
client identifiers. For each distinct query identifier in set U, a MATCH
SOURCE$(id(S), id(q), l(r))$ message is created and forwarded to the ap
client. Finally S removes $head(L)$ from L and U from T, and forwards C
FICATION 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
super-peer to store the continuous queries it is responsible for. These
structures are utilised in the step of DHTrie described in Section 3.4 to
ciently which queries match a given publication.

4.1 The Algorithm BestFitTrie

To index queries BestFitTrie utilises an array, called the *attribute directo*
that stores pointers to word directories. *AD* has one element for each di
tribute in the query database. For a query of the form presented in Secti



(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 a roots of *tries* in a *forest* that is used to organize *sets of words* – the set in wp_i for each atomic formula $A_i \sqsupseteq wp_i$ in a query (denoted by *word* The proximity formulas contained in each wp_i are stored in an array c *proximity array* (PA). PA stores pointers to trie nodes (words) that are in proximity formulas along with the respective proximity intervals for mula. There is also a hash table, called *equality table* (ET), that indexe values s_j, $1 \leq j \leq m$ that appear in atomic formulas of the form $A_j = s_j$. T idea behind BestFitTrie is to identify *common parts* between queries (thr use of identifying subsets; an example is shown in Table 1) and to expl commonalities to achieve faster filtering. BestFitTrie is described in detai

To evaluate the performance of BestFitTrie we have also implement rithms BF, SWIN and PrefixTrie. BF (Brute Force) has no indexing stra scans the query database sequentially to determine matching queries. SW gle Word INdex) utilises a two-level index for accessing queries in an way. A query of the form presented at Section 3.2, is indexed by SWIN its attributes $A_1, \ldots, A_n$ and also under m text values $s_1, \ldots, s_m$ and $n-$ selected randomly from $wp_m + 1, \ldots, wp_n$. More specifically SWIN utilise to index equalities and an *AD* pointing to several *WD*s to index the ator tainment queries. Atomic queries within a *WD* slot are stored in a list. P is an extension of the algorithm Tree of [14] appropriately modified to c attributes and proximity information. Tree was originally proposed fo *conjunctions of keywords* in secondary storage in the context of the SD SIFT. Following Tree, PrefixTrie uses *sequences* of words sorted in lexic order for capturing the words appearing in the word patterns of atomi las (instead of sets used by BestFitTrie). A trie is then used to store se compactly by exploiting *common prefixes* [14].

Algorithm BestFitTrie constitutes an improvement over PrefixTrie.
PrefixTrie examines only the prefixes of sequences of words in lexicogr
der to identify common parts, it misses many opportunities for cluste
Figure 1(b) where the trie constructed by PrefixTrie for the example q
Table 1 is shown). BestFitTrie keeps the main idea behind PrefixTrie bu
dles the words contained in a query as a set rather than as a sorted sequ
(ii) searches exhaustively the forest of tries to discover the best place to i
a new set of words. This allows BestFitTrie to achieve better clustering a
in Figure 1, where we can see that it needs only one trie to store the set
for the formulas of Table 1, whereas PrefixTrie introduces redundant n
are the result of using a lexicographic order to identify common parts. T
redundancy can be the cause of deceleration of the filtering process as
show in the next section. To improve beyond BestFitTrie it would be in
to consider *re-organizing* the word directory every time a new set of w
rives, or periodically, but this might turn out to be prohibitively expensiv
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 se
uments downloaded from ResearchIndex[3] and originally compiled in
documents are research papers in the area of Neural Networks and we w
to them as the NN corpus. Because no database of queries was availab
we developed a methodology for creating user queries using *words* and *t*
terms (phrases) extracted automatically from the Research Index docume
the C-value/NC-value approach of [5].

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

The first experiment that we conducted to evaluate BestFitTrie targ
performance under different sizes of the query database. In this experi
randomly selected one hundred documents from the NN corpus and us
as incoming documents in query databases of different sizes. The size
matching percentage for each document used was different, but the avera
ment size was 6869 words, whereas on average 1% of the queries stored
the incoming documents.

As we can see in Figure 2, the time taken by each algorithm grows
with the size of the query database. However SWIN, PrefixTrie and Be
are less sensitive than Brute Force to changes in the query database s

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

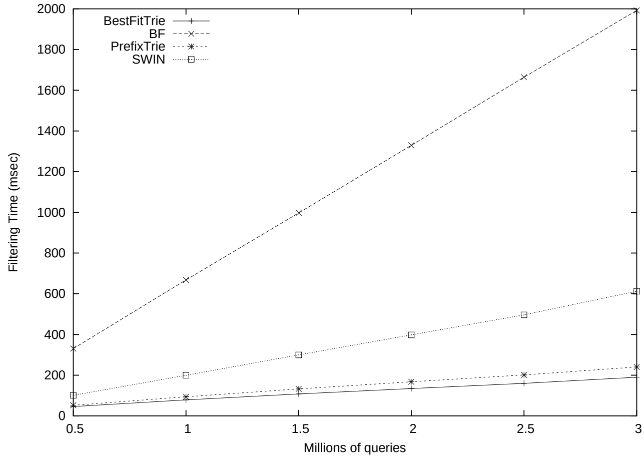


Figure 2: Effect of the query database size in filtering time

trie-based algorithms outperform SWIN mainly due to the clustering te
that allows the exclusion of more non-matching atomic queries during
We can also observe that the better exploitation of the commonalities
queries improves the performance of BestFitTrie over PrefixTrie, resul
significant speedup in filtering time for *large query databases* (BestFitTri
faster than PrefixTrie and 1000% faster than sequential scan for a datab
million queries). Additionally, Figure 3 contrasts the algorithms in terms
insertion time. In this figure we can see the average time in millisecond
to insert a new query in a query database of different sizes. Notice that in
of 2.5 millions of queries BestFitTrie needs 5 milliseconds more to inser
into the database, to save about 45 milliseconds at filtering time.

Comparison of the algorithms in terms of throughput results in Be
giving the best filtering performance managing to process a load of abou
(about 9 ResearchIndex papers) per second for a query database of 3
queries.

In terms of space requirements BF needs about 15% less space than
based algorithms, due to the simple data structure that poses small space
ments. Additionally the rate of increase for the two trie-based algorithms i
to that of BF, requiring a fixed amount of extra space each time. From th
ments above it is clear that BestFitTrie speeds up the filtering process wit
extra storage cost, and proves faster than the rest of the algorithms, man
filter as much as 3 million queries in less the 200 milliseconds, which is
times faster than the sequential scan method.

We have also evaluated the performance of the algorithms under tw
parameters: *document size* and *percentage of queries matching a publish*

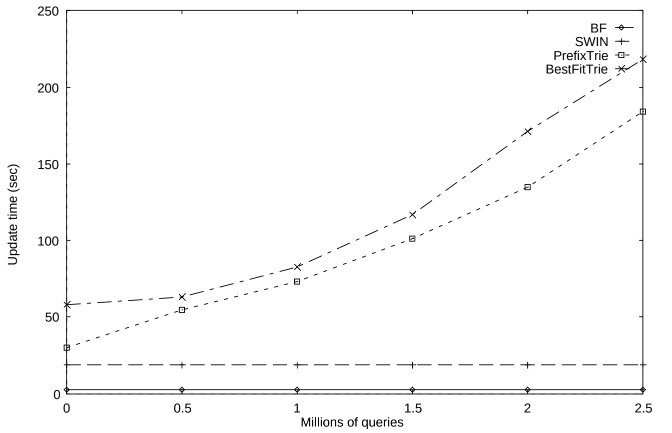


Figure 3: Query insertion time for different query database sizes

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

Finally we have developed various heuristics for ordering words in maintained by PrefixTrie and BestFitTrie when *word frequency* inform *word ranking*) is available, as it is common in IR research [2]. Using th *rank* heuristic (*irank*) [14], we store the least frequent words of the que the roots of the tries, while the frequent ones are pushed deeper in the sulting in many narrow tries. Thus more queries are put in subtrees occurring less frequently, resulting in less lookups during filtering time gorithms using the irank heuristic are PrefixTrie-irank and BestFitTrie-ir faster algorithm is shown to be a variation of BestFitTrie, called LCWTr Common Word), where BestFitTrie is limited to consider a single candi during query insertion: the one that has the least frequent word of the atom as root. The details of the experiments briefly mentioned above are pres detail in [13].

5 Work in Progress

Performance evaluation in the distributed case. To evaluate the perf and scalability of DHTrie we are currently implementing the algorithms

tion 3. We plan to evaluate DHTrie by considering its behaviour (mainly e
in terms of message load between super-peers) under various paramete
and resource size, arrival rates of queries and resources, number of sup
etc.). Furthermore we plan to extend the algorithm to take locality iss
account by maintaining extra routing tables for the most frequently c
supers-peers (namely the super-peers responsible for the most freque
published).

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

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

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

Word frequency computation in a distributed setting. Computing the f
of occurrence of words in a distributed setting is a crucial problem if o
to support vector space queries or to provide for load balancing among
showed above. There are mainly two approaches to the word frequency
tation in a distributed setting; (a) a global ranking scheme that assumes
authority that maintains the frequency information or a message-intensiv
mechanism that notifies every peer in the network about changes in frequ
formation or (b) a local ranking scheme that computes word frequencie
p_i based solely on frequencies of words in documents that are publishe
is clear that the first approach affects the scalability and efficiency of th
while the second approach can be misleading due to peer specialisation.

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

global "truth" through an updating and estimation mechanism.
Reducing network traffic. We can reduce network traffic by *compressi*
cations. In the full version of the paper we describe a gap compression t
that allows the matching of a compressed publication against a databas
queries using algorithm BestFitTrie.

Christos Tryfonopoulos is with the Dept. of Electronic and Computer Engineeri
nical University of Crete, 73100 Chania, Crete, Greece. E-mail: trifon@intellige

Manolis Koubarakis is with the Dept. of Electronic and Computer Engineerin
cal University of Crete, 73100 Chania, Crete, Greece. E-mail: manolis@intellige

References

[1] ABERER, K., DATTA, A., AND HAUSWIRTH, M. Multifaceted Si
ous Load Balancing in DHT-based P2P systems: A new game with
and bins. Tech. Rep. IC/2004/23, Swiss Federal Institute of Te
Lausanne (EPFL), 2004.

[2] BAEZA-YATES, R., AND RIBEIRO-NETO, B. *Modern Informa
trieval*. Addison Wesley, 1999.

[3] CARZANIGA, A., ROSENBLUM, D.-S., AND WOLF, A. Design a
uation of a wide-area event notification service. *ACM TOCS 19*, 3
2001), 332–383.

[4] CHANG, C.-C. K., GARCIA-MOLINA, H., AND PAEPCKE, A.
Rewriting for Translating Boolean Queries in a Heterogeneous Inf
System. *ACM TOIS 17*, 1 (1999), 1–39.

[5] DONG, L. Automatic term extraction and similarity assessment in a
specific document corpus. Master's thesis, Dept. of Computer Scie
housie University, Halifax, Canada, 2002.

[6] IDREOS, S., KOUBARAKIS, M., AND TRYFONOPOULOS, C. P2
Ad-hoc and Continuous Queries in Super-peer Networks. In *Proce
EDBT 2004* (Heraklion, Crete, Greece, 14–18 March 2004).

[7] IDREOS, S., TRYFONOPOULOS, C., KOUBARAKIS, M., AND D
Y. Query Processing in Super-Peer Networks with Languages B
Information Retrieval: the P2P-DIET Approach. In *Proceedings o
DB 2004* (2004).

[8] KOUBARAKIS, M., TRYFONOPOULOS, C., IDREOS, S., AND D
Y. Selective Information Dissemination in P2P Networks: Probl
Solutions. *ACM SIGMOD Record, Special issue on Peer-to-Peer D
agement, K. Aberer (editor) 32*, 3 (September 2003).

[9] KOUBARAKIS, M., TRYFONOPOULOS, C., RAFTOPOULOU, KOUTRIS, T. Data models and languages for agent-based textual tion dissemination. In *Proceedings of CIA 2002* (September 2002), v of *LNCS*, pp. 179–193.

[10] PIETZUCH, P., AND BACON, J. Peer-to-Peer Overlay Broker Net an Event-Based Middleware. In *Proceedings of DEBS'03* (June 20

[11] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M., AND B ISHNAN, H. Chord: A Scalable Peer-to-peer Lookup Service for Applications. In *Proceedings of the ACM SIGCOMM '01 Confere* Diego, California, August 2001).

[12] TERPSTRA, W., BEHNEL, S., FIEGE, L., ZEIDLER, A., AND BUC A. A Peer-to-Peer Approach to Content-Based Publish/Subscribe. *ceedings of DEBS'03* (June 2003).

[13] TRYFONOPOULOS, C., KOUBARAKIS, M., AND DROUGAS, Y. Algorithms for Information Retrieval Models with Named Attrib Proximity Operators. Submitted to a conference. Under review, 20

[14] T.W. YAN AND H. GARCIA-MOLINA. Index structures for selec semination of information under the boolean model. *ACM TOL* (1994), 332–364.

[15] ZIPF, G. K. *Human Behaviour and Principle of Least Effort*. Wesley, Cambridge, Massachusetts, 1949.