
Pub/Sub Functionality in IR Environments using Structured Overlay Networks

Christos Tryfonopoulos, Stratos Idreos and Manolis Koubarakis

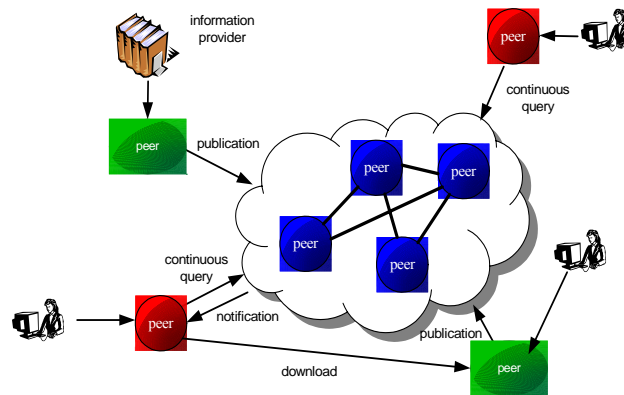
Intelligent Systems Laboratory
Dept. of Electronic and Computer Engineering
Technical University of Crete
<http://www.intelligence.tuc.gr/>

Outline

- Motivation
 - Related work
 - Background
 - The DHTrie protocols
 - Protocol presentation
 - Experimental evaluation
 - Open problems
-

Motivation

■ Publish/subscribe scenario



Related Work

■ Information Retrieval (in P2P networks)

- Unstructured P2P networks
 - PlanetP, PIRS, [KJ04], [YF05], Peerware
- Hierarchical P2P networks
 - [LC03], [LC05], P2P-DIET, LibraRing, Odissea
- Structured P2P networks
 - pSearch, Arpeggio, Minerva, Galanx

Related Work

- Information Filtering
 - Centralised
 - InRoute, SIFT, DIAS
 - Hierarchical P2P networks
 - P2P-DIET
 - Structured P2P networks
 - pFilter
 - Content-based pub/sub on top of structured P2P networks
 - Meghdoot, Hermes, [TBF+03], [TA04]

Background: data model

- **Publication:** a set of attribute-value pairs with at most one value per attribute.
- Example:

(*AUTHOR* = “Manolis Koubarakis”,
TITLE = “Peer-to-Peer Publish/Subscribe Networks with
Languages from Information Retrieval”,
ABSTRACT = “We study ...”)

Background: query language

- Interpret text values under the **boolean** or **vector space model**, depending on their use in queries.
- QL: Conjunctions of **attribute-operator-value** atomic formulas (atomic queries). Example:

$(AUTHOR = "John\ Brown") \wedge$
 $(TITLE \delta (algorithms \prec_{[0,2]} complexity) \wedge filtering) \wedge$
 $(ABSTRACT \sim_{0.6} "Information\ alert\ in\ structured\ P2P...")$

- Extendable with other data types and richer structure.

Distributed Hash Tables (DHTs)

- Second generation structured overlay networks.
- Created to solve the **object location** problem in a distributed and dynamic network of nodes:

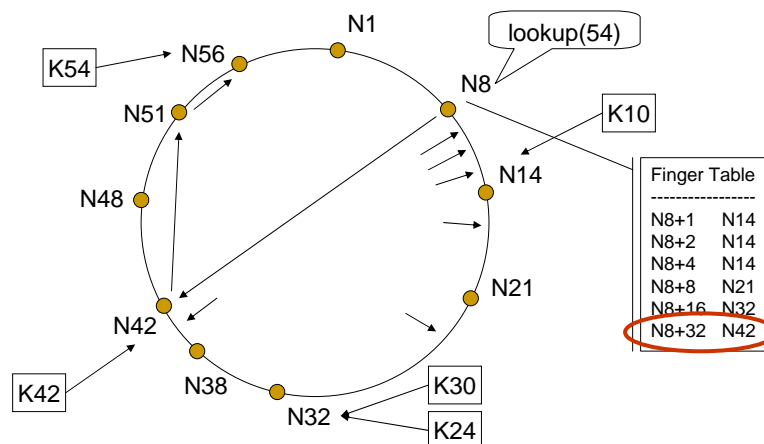
Let K be some data item. Find K!

- Distributed version of **hash table** data structure.
- Main operations:
 - **Put**: given a key (for a data item), map the key onto a node.
 - **Get**: Find the location of a data item with a given a key.

Chord

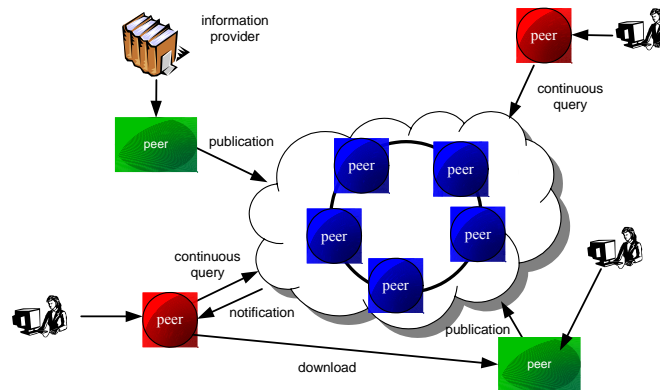
- Node IDs: m bits (e.g., produced using SHA-1 on IPs)
- Nodes organised in a **circle** in the ID space.
- Keys are hashed using the same hash function and assigned to the **successor** node in the ID circle.
- For correct routing, nodes need **only know their successor**. For better routing, Chord maintains a routing table with m entries (**fingers**) pointing to nodes **spaced exponentially** in the ID space.
- We can locate a node in $\log(N)$ hops, where N is the number of nodes.

The Chord Ring



The DHTrie protocols

- Publish/subscribe scenario revisited



The DHTrie protocols

- Basic idea: **Index and store** subscriptions in the DHT. Make sure publications **meet** subscriptions.
- Two levels of indexing:
 - **Global**: Use an extension of a DHT.
 - **Local**: Use what is appropriate for your subscription language. In our case a trie-like structure (BestFitTrie), and other bits.

The DHTrie protocols (cont'd)

- Subscribing with **Boolean** atomic queries

Assume query q of the form:

$$(A_1 = s_1) \wedge \dots \wedge (A_m = s_m) \wedge (A_{m+1} \tilde{\circ} wp_{m+1}) \wedge \dots \wedge (A_n \tilde{\circ} wp_n)$$

1. Select a single word w contained in any of s_i or wp_j .
2. Index query in node **responsible** for $id=H(w)$.

The DHTrie protocols (cont'd)

- Subscribing with **vector space** atomic queries

Assume query q of the form:

$$(A_1 \sim_{k_1} s_1) \wedge \dots \wedge (A_n \sim_{k_n} s_n)$$

1. Construct D_1, \dots, D_n , the sets of distinct words in s_1, \dots, s_n .
2. Construct list $L = \{H(w_j) : w_j \in D_1 \cup \dots \cup D_n\}$
3. Remove duplicates from L .
4. Index query in **all nodes** responsible for an $id=H(w_j)$.

The DH'Trie protocols (cont'd)

- Subscribing with **mixed type** queries

Assume query q of the form:

$$(A_1 = s_1) \wedge \dots \wedge (A_m = s_m) \wedge \\ (A_{m+1} \tilde{\text{d}} wp_{m+1}) \wedge \dots \wedge (A_n \tilde{\text{d}} wp_n) \wedge \\ (A_{n+1} \sim_{a_{n+1}} s_{n+1}) \wedge \dots \wedge (A_k \sim_{a_k} s_k)$$

- Index q under its Boolean part only

The DH'Trie protocols (cont'd)

- Publishing a resource

Assume a publication p of the form:

$$\{(A_1, s_1), (A_2, s_2), \dots, (A_n, s_n)\}$$

1. Construct D_1, \dots, D_n , the sets of distinct words in s_1, \dots, s_n .
2. Construct list $L = \{H(w_j) : w_j \in D_1 \cup \dots \cup D_n\}$
3. Remove duplicates from L .
4. Forward publication in **all** nodes responsible for an $id=H(w_j)$.
5. Nodes will find all matching queries using their local data structures and algorithms and notify subscribers.

The DH Trie protocols (cont'd)

- Notifying interested subscribers
 - To find all matching queries, nodes utilize a **local (centralised) algorithm** which combines:
 - BestFitTrie (Boolean part)
 - SQI (VSM part)
 - Any other efficient centralised algorithm could be used.
 - Notifications are sent to:
 - the IP address associated with the query (nodes that are online).
 - the successor of the recipient (nodes that are offline).
 - Notifications could be batched and sent when network traffic is low depending on the application.

The DH Trie protocols (cont'd)

- Other protocols for pub/sub scenario:
 - Removing a query
 - Updating a query
 - Joining or leaving the network
- Also designed protocols to support one-time queries (IR).
 - See LibraRing system for distributed DLs. (ECDL'05)

Message routing

- **Boolean query indexing** can be handled in a straightforward way using Chord's **put** primitive.
 - **Resource publication** and **VSM queries** require sending the same message to **a group** of nodes.
 - Multicasting techniques are **not applicable**, since the group of nodes to be contacted is not known a priori.
-

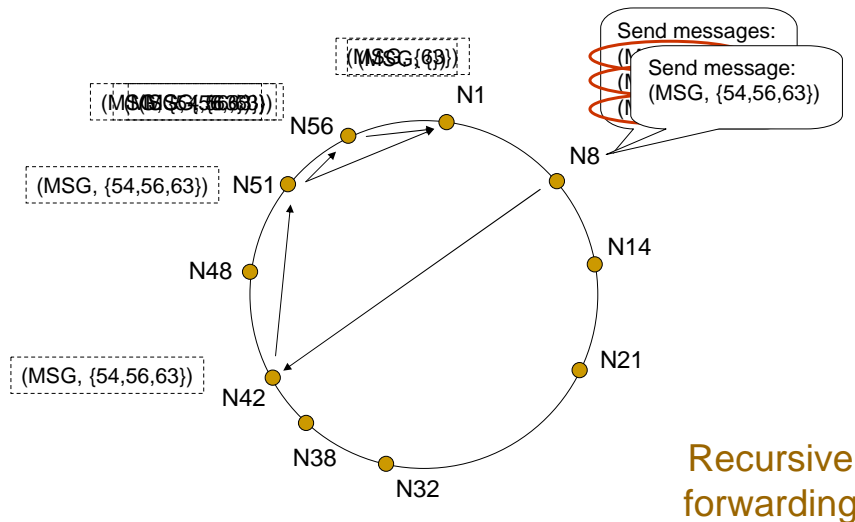
Resource publication

- Two methods: *iterative vs. recursive*
 - *Iterative*: send a *publish()* message to each recipient.
 - For k recipients we need $O(k \log(N))$ publish messages.
-

Resource publication (cont'd)

- *Recursive* message forwarding
- Method: Node P wants to contact a group of nodes with ids: $L = \{id_1, \dots, id_n\}$:
 1. Sort ids in ascending order.
 2. Constructs message (msg, L) and sends it to node responsible for $head(L)$ - that is peer P' with $id(P') \leq head(L)$
 3. Upon reception by P' , $head(L)$ is checked
 - i. If $id(P') < head(L)$, then P' forwards the message as above.
 - ii. If $id(P') \geq head(L)$, then P' copies msg and deletes $head(L)$.

Iterative vs Recursive forwarding



Frequency Cache (FCache)

- Extra *routing table*.
- Stores IPs for nodes responsible for queries containing *frequent words*
 - reach nodes we contact often in 1 hop
- No extra message cost:
 - Uses only *local* information (published documents).
 - Contact information discovered when needed.
- FCache misses:
 - Mainly for infrequent words.
 - Cost a single *lookup()* operation (needed anyway ...).

Implemented Algorithms

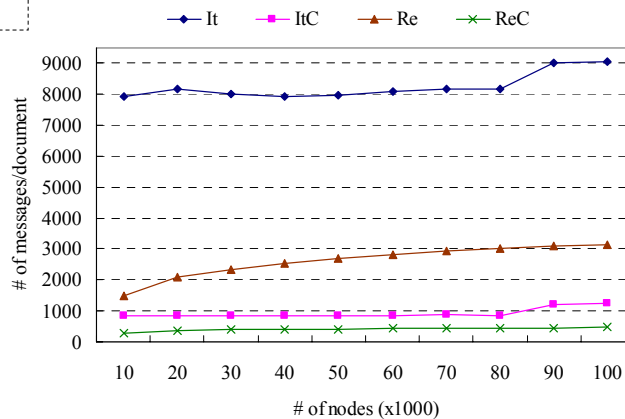
Algorithm name	Iterative / Recursive	FCache utilisation
It	Iterative	✗
ItC	Iterative	✓
Re	Recursive	✗
ReC	Recursive	✓

Experimental Evaluation

- Evaluated the algorithms under different parameters:
 - Network size
 - Query types
 - FCache size & level of training
 - Publication size
- Also looked into the load balancing problem (load shedding):
 - Filtering load
 - Routing load

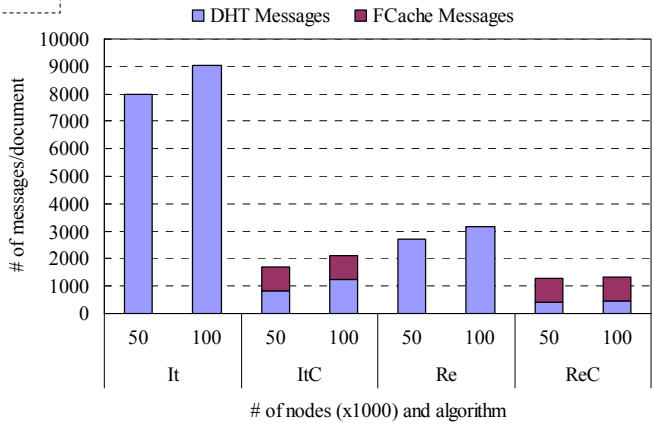
Experimental Evaluation

Performance for various network sizes



Experimental Evaluation

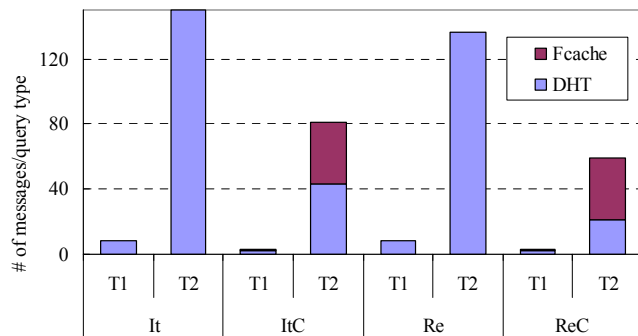
Total number of messages for a single document



Experimental Evaluation

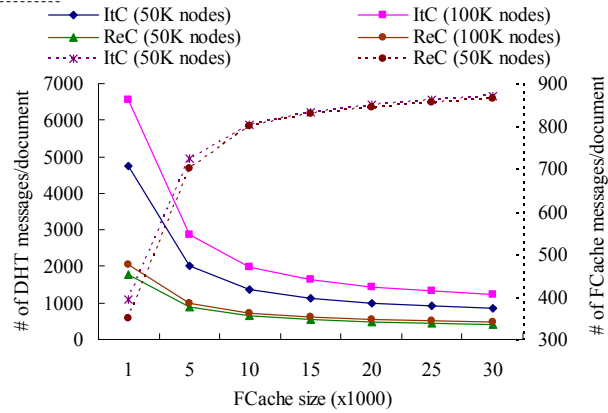
Total number of messages for indexing a query of different types

T1: Boolean or mixed type queries
T2: Vector space queries



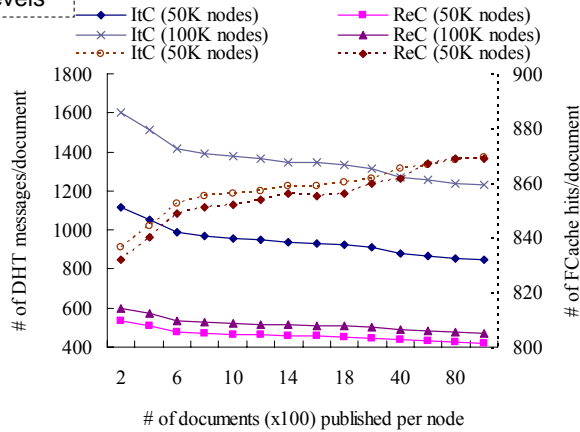
Experimental Evaluation

Performance for different FCache sizes



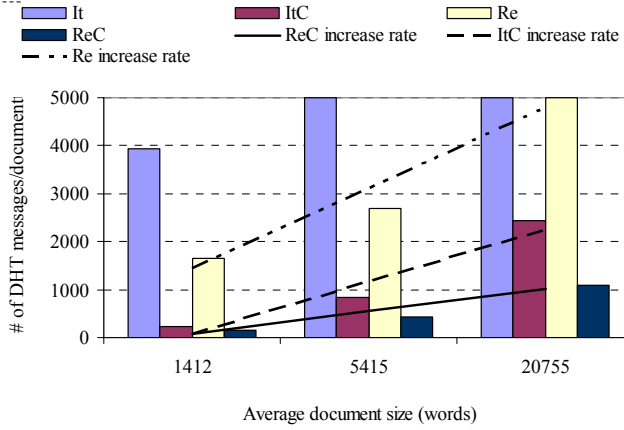
Experimental Evaluation

Performance for various FCache training levels



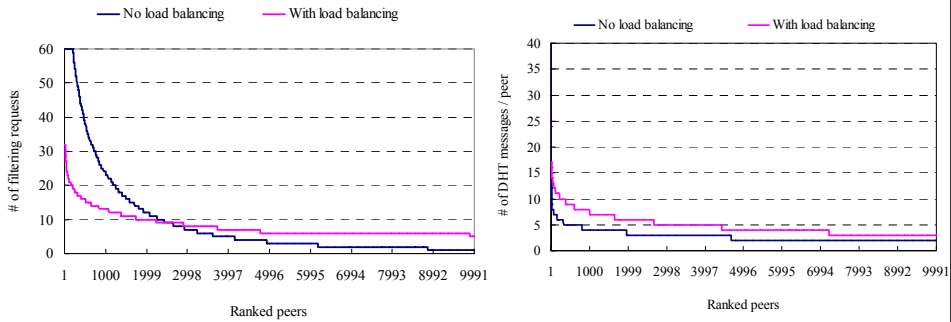
Experimental Evaluation

Performance for different document sizes



Experimental Evaluation

Filtering load and routing requests for the top 10K nodes.



Open Problems

- System development and deployment on PlanetLab.
 - Distributed computation of word frequencies.
 - Load balancing.
 - More expressive data models (XPath/XQuery with full text).
-

Acknowledgements - Funding

- Evergrow <http://www.evergrow.org> (IST/FET IP on Complex Systems).
 - Heraclitus (Greek Government PhD Fellowship Programme).
-