Distributed Resource Sharing using Self-Organized Peer-to-Peer Networks and Languages from Information Retrieval[*]

Christos Tryfonopoulos[†] Manolis Koubarakis

Dept. of Electronic and Computer Engineering,
Technical University of Crete, 73100 Chania, Crete, Greece
{trifon,manolis}@intelligence.tuc.gr

Abstract

A useful computational problem that can be solved effectively and efficiently through self-organization techniques is distributed resource sharing in the Web. The architecture we envision supports both query and publish/subscribe functionality using languages from Information Retrieval. We propose to approach this problem using ideas from self-organized peer-to-peer networks and especially distributed hash tables like Chord. This paper concentrates only on how to offer the envisaged publish/subscribe functionality and discusses our current results.

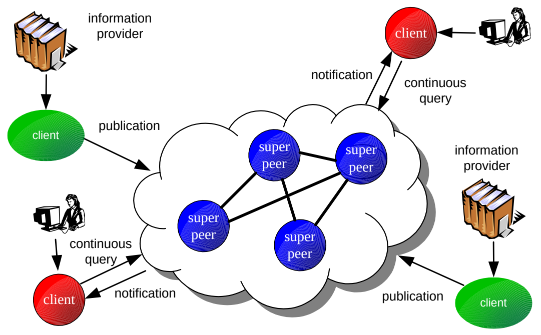

Figure 1: An architecture for distributed resource shring

1 Introduction

A useful computational problem that can be solved effectively and efficiently through self-organization techniques is *distributed resource sharing* in the Web. Figure 1 presents an architecture for distributed resource sharing which has been adopted in our previous work [5, 3, 7, 2]. There are two kinds of basic functionality that we expect the architecture of Figure 1 to offer:

- *One-time querying*: a user utilizes his *client* to pose a *query* (e.g., "I want papers on self-organization") and the system returns a list of pointers to matching resources owned by other clients in the network.
- *Publish/subscribe (pub/sub)*: In a pub/sub scenario, a user posts a *continuous query* to the system to receive a notification whenever certain *events* of interest take place (e.g., when a paper on self-organization becomes available).

In this paper we show how to solve the problem of building pub/sub functionality in the architecture of Figure 1 using ideas from self-organized distributed hash tables (DHTs) like Chord [6].

In our architecture we have 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.

We will use the the data model $\mathcal{AWP}$ inspired from Information Retrieval for specifying queries and resource metadata [5]. For the purposes of this paper, a *(resource) publication* is a set of attribute-value pairs (A, s) where A is a *named attribute*, s is a *text* value and all attributes are *distinct*. The

[*]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.

[†]The author is partially supported by a Ph.D. fellowship from the program Heraclitus of the Greek Ministry of Education.

following is an example of a publication:

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

The query language of $\mathcal{AWP}$ offers *Boolean* and *word proximity operators* on attribute values. The following is an example of a conjunctive $\mathcal{AWP}$ query:

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

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

2 The DHTrie protocols

We implement a distributed resource sharing by a set of protocols called the *DHTrie protocols* (from the words DHT and trie). The DHTrie protocols use *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 DHT 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 *common words* in atomic queries is utilised.

2.1 Self-organisation in the Chord ring

We use a Chord-like DHT to implement our super-peer network. Chord [6] uses consistent hashing to map keys to nodes. Each node and data item is assigned an k-bit identifier, where k 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.* Data items and nodes self-organize in the Chord ring as follows. A new data item r is stored at the node with identifier $H(r)$ if this node exists, given that H is the hash function used. 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. Node identifiers are assigned to nodes by hashing their respective IP addresses using a cryptographic hash function. When a node joins the Chord ring, its predecessor finds out that a new node has joined and makes this node responsible for data items hashing in identifiers between itself and the new node. Discovering that a node has joined is achieved through a self-stabilisation protocol that every node runs periodically. The idea behind self-stabilisation is to keep a node informed about its immediate successor in the Chord ring.

2.2 Subscribing with a continuous query

Let us assume that a client C wants to submit a continuous query q of the form $A_1 = s_1 \wedge ... \wedge A_m = s_m \wedge A_{m+1} \sqsupseteq wp_{m+1} \wedge ... \wedge A_n \sqsupseteq wp_n$. C contacts a super-peer S (its *access point*) and sends it a message SUBMITCQUERY$(id(C), q)$, 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 message FWDCQUERY$(id(S), id(q), q)$ 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 message FWDCQUERY containing q, it inserts q in its local data structures using the insertion algorithm of BestFitTrie described briefly in [7, 4].

2.3 Publishing a resource

When client C wants to publish a resource, it constructs a publication p of the form $\{(A_1, s_1), (A_2, s_2), \ldots, (A_n, s_n)\}$, it contacts a super-peer S and sends S a message PUBREsource$(id(C), p)$. 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 message FwdResource$(id(S), id(p), p, L)$, 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 message FwdResource 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 2.4). 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.

2.4 Notifying interested subscribers

Let us now examine how notifications about published resources are sent to interested subscribers. When a message FwdResource 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 presented in [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 Section 2.3.

Upon arrival of a message CQNotification 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 2.3. 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 message MatchingResource$(id(S), id(q), l(r))$ is created and forwarded to the appropriate client. Finally S removes $head(L)$ from L and U from T, and forwards message CQNotification according to the algorithm described in Section 2.3.

We are currently implementing the protocols described in Sections 2.2-2.4 to evaluate their performance and scalability. The results of the evaluation will be presented at the workshop.

3 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.

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 [9]). 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 [1]. 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 [7, 4] where queries are indexed under infrequent words; we also use a form of controlled replication to deal with overloading due to notification processing.

4 Word frequency computation in a distributed setting

Computing the frequency of occurrence of words in a distributed setting is a crucial problem for query processing[1] and for providing load balancing as shown 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.

We are currently developing a distributed word ranking algorithm that is a hybrid form of the two approaches described earlier. We provide an algorithm that is based on local information, but it also tries to combine this information with the global "truth" through an updating and estimation mechanism.

5 Conclusions

We presented our current work on the problem of distributed resource sharing using self-organised P2P networks and languages from Information Retrieval.

References

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, 1999.

[2] 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.

[3] 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)*, pages 851–853, Heraklion, Crete, Greece, 14–18 March 2004.

[4] S. Idreos, C. Tryfonopoulos, M. Koubarakis, and Y. Drougas. Query Processing in Super-Peer Networks with Languages Based on Information Retrieval: the P2P-DIET Approach. In *Proceedings of P2P & DB 2004*, 2004.

[5] M. Koubarakis and T. Koutris and P. Raftopoulou and C. Tryfonopoulos. 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.

[6] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[7] C. Tryfonopoulos, M. Koubarakis, and Y. Drougas. Filtering Algorithms for Information Retrieval Models with Named Attributes and Proximity Operators. In *Proceedings of the 27th Annual ACM SIGIR Conference*, Sheffield, United Kingdom, July 25-July 29 2004. Forthcoming.

[8] T.W. Yan and H. Garcia-Molina. The SIFT information dissemination system. *ACM Transactions on Database Systems*, 24(4):529–565, 1999.

[9] G. K. Zipf. *Human Behaviour and Principle of Least Effort.* Addison Wesley, Cambridge, Massachusetts, 1949.

[1]Word frequencies are useful for processing queries in $\mathcal{AWP}$ [7] but also vector space queries [8].