Query Routing in P2P Networks

Marc Cromme
Indexdata
Købmagergade 43
Denmark
marc@indexdata.dk

2004-06-07

Contents

1 Structured Queries

1.1 CCL etc.

Candidate structured query language include simple, intuitive but less powerful languages such CCL (the Common Command Language, ISO 8777) and the query languages used by popular web-search engines such as google.com's. However, these lack the rigour and expressiveness that we would like the Alvis system to be able to support.

1.2 CQL

CQL stands for Common Query Language. It is a formal language for representing queries to Information Retrieval systems such as as web indexes, bibliographic catalogues and museum collection information. It is being developed by the Z39.50 Maintenance Agency as part of its ZING initiative ("Z39.50-International: Next Generation").

The interesting part of the CQL language in this context are its boolean operators and index relations. These features decompose nicely into set algebraic features.

1.2.1 CQL boolean operators

Queries may be joined together using the four boolean operators, `and`, `or`, `not` and `prox`.

and set intersection, both terms must ocurr in a record, like `dinosaur and reptile`

or set union, only one of the terms must ocurr in a record, like `dinosaur or reptile`

not in CQL is a boolean operator, really ANDNOT, like `dinosaur not reptile`. The language is defined this way in order to prevent the formulation of set complementation queries that many engines cannot execute efficiently, like `not reptile`. However, for algebraic purposes, we can consider CQL to support unary negation, expressed as a universal search combined with an ANDNOT.

prox this is a special kind of AND which requires its operands to occur close to each other, like `dinosaur prox reptile`. This finds any record containing the string `dinosaur reptile` or the string `reptile dinosaur`.

In the general case, the proximity operator's semantics are affected by four parameters, which are expressed in the general form

```
prox/distance RELATION DISTANCE/unit=UNIT/(ordered|unordered)
```

In detail,

relation indicates whether the operands should be separated by an exact distance, a distance greater than or equal to that specified, etc. It defaults to <= if not specified explicitly.

distance specifies the number of units (words, sentences, etc.) that may separate the operands, with zero indicating the same unit and one indicating adjacent units. The default is 1 if the unit is word, 0 otherwise.

unit specifies whether the proximity condition is about how many words, sentences, paragraphs or elements separate the operands. It defaults to `word`.

ordering may be `ordered` to specify that the operands must occur in the record in the same order as they do in the query, or `unordered` if they may occur in either order (which is the default).

Some examples on these boolean operators are quite in order to illustrate their behaviour:

`foo prox bar` The words foo and bar immediately adjacent to each other, in either order.

`foo prox///sentence bar` The words foo and bar ocurring anywhere in the same sentence. (Recall that the default distance is zero when the unit is not word.)

`foo prox//3/element bar` The words must occur within three elements of each other: for example, if a record contains a list of authors, and author number 4 contains foo and author number 7 contains bar then this search will find that record.

`foo prox/=/2/paragraph bar` The words must appear exactly two paragraphs apart: it is not good enough for them to appear in the same paragraph or in adjacent paragraphs.

`foo prox/>/4/word/ordered bar` Finds records in which the words appear, with foo first, followed more than four words later by bar, in that order.

1.3 CQL Parsers

One of the attractions of CQL is that open source CQL parsers are available in several languages, including C, Java and Python.

2 Query routing

Distributed hash tables (DHTs) and vector space-based similarity measures are not the only solutions to the problem of finding a peer that holds good documents in relation to a query, and retrieving the hit set of documents from the peer network.

Here we propose a query routing approach that directs queries to peers that are likely to contain relevant response documents. Essentially, we propose to store query routing information, not document indexes, either in a network global Distributed Query Routing Table (DQRT), or more lightweight, in Local Query Routing Tables (LQRTs).

2.1 Distributed Hash Tables

Distributed hash tables are by definition a way to distribute some part of the content of documents across the P2P network, thus making it feasible that queries ask the peer network about documents.

The essential idea of DHTs is that peer-IDs and data-IDs can be mapped to the same space, usually by a checksum algorithm like MD5. There must be some notion of 'distance' between the thing you want to find, and the place where it is stored - whether the thing being sought is the document itself or a proxy or digest containting information about where to find the document.

DHTs are usually about locating one thing, namely a document with a given key.

One assumption of DHT based querying is that there is a common model of what documents look like, and which things - or features of documents - are deemed important and therefore distributed.

In many cases (e.g. in Gnutella), the name of the document file is the only feature which is distributed - an approach which clearly does not suffice in an distributed information retrieval system.

Alvis must do better to support more complex features or queries in the network: given a query, it must find a set of documents satisfying the query, and it must rank them over all hit sets from all peers contacted.

The problem is that the key of the document will not help at all, because it can not be constructed from the query. That's where the train jumps off the rails using out-of-the-box DHTs.

Likewise, the hash key of a query (interpreted as a small document) does not relate to the hash key of the atomic parts of the documents. The network can not make an association between query and document this way either.

As far as I see, it is only possible and interesting to give the atomic parts of queries a key, that is, the 'terms' of CQL, or the 'words' of 'bag-of-word' queries, and to match parts of queries with documents that might satisfy them.

Thus, the best a DHT based approach can do is to build a global distributed inverse index over document terms, split a query into its atomic parts, use the keys of the atoms to identify the peers which do know about documents containing this atomic part, using the distributed inverted indexes.

Then, having several lists of all documents satisfying some atomic part, the requesting peer has to compute the hit set list. Finally, a network-global document retrieval of all documents in the hit set must be made to merge and rank.

This just will not fly: our Zebra machine can search 20 million indexed documents with boolean, structured queries in less than one second of search time, only because there are carefully laid out indexes on the disk, and partially in memory. Searching 20 million DHT inverse indexed documents - taking into account network latency just to look up indexes - is just not feasible within hours or days of search time.

Global distributed inverted indexes over document terms are simply not feasible. Period.

Another assumption with regular DHTs is that parts of the documents are legally distributable across the peer network (and maybe even legally replicated).

All these assumptions are not necessarily met in the use case 'Digital Library System', and an alternative to DHTs might be necessary.

2.2 Local indexing resources

A better approach is to forward queries to peers, and let the peers use their own, local and fast indexing facilities to boil the queries down to some small hit set. Finally, only the most relevant parts of the hit set are shipped to the querying peer - which will be merging the partial, short hit sets to a pseudo-global hit set.

This way, network latency is saved, since no peer network-global inverted document term indexes have to be updated and maintained.

The question is simply how to make sure that all relevant peers have been queried? The short answer is: there can be no such guarantee in large, non-trivial networks. However, techniques can be used to ensure that it is increasingly probable that an increasingly large proportion of relevant peers receive the query.

2.3 Query flooding

The simplest approach is to flood the network with a given query, allowing each peer to see and answer the query, if it holds relevant documents. This approach does not scale well with the number of peers, and will soon break down.

Or, one could only flood a local neighborhood of a given peer, which scales well. The disadvantage is that the query does not reach all nodes, and might remain unanswered by the best possible peer. Local flooding does not make is probable that relevant peers are queried.

2.4 Distributed Query Routing Table

Having given up total assurance that we can get the 'true' hit set to a query in a peer network, the next best is a probability assurance that we have at least hit the most important peers in relation to a query.

For this, we need query routing tables in peer-to-peer networks. One can balance between the advantages of locality and total flooding by keeping routing information for queries. This is ideally done such that a path a query takes to some peers scales like $r \log(n)$, where r is the number of wanted parallel peer hits of a query, and n is the peer count of the network. Then r is set high enough to satisfy the probability that relevant peers have been hit by the query, but low enough to prevent swamping of the network.

One could turn the DHT approach around, assuming that the only thing all peers have in common is the ability to understand a particular query syntax or form, and distribute key features of successfully answered queries over the network. This distributed table over parts of queries forms a Distributed Query Routing Table (DQRT), assuring that queries hit - with high probability - the peers that can answer them sucessfully.

2.5 Local Query Routing Table

One could even go one step further, and not address the global properties of such a query routing approach, but rely on good local query routing properties only. This way, one has not to impose the burden on the network to maintain a globally distributed structure.

The price to pay for only applying Local Query Routing Tables (LQRT) is of course that there is no guarantee that a query will hit the globally best suited peer to retrive the document. However, if constructed properly, it will hit a set of good peers holding documents of sufficent relevance to the query.

2.6 Adaptive query routing table update

Our proposal is that the DQRT or LQRT is built on the fly using the information which under all circumstances has to be passed in the network: the queries and query responses.

It is assumed that the query model has atomic parts, which can be used for index information. In the case of 'bag-of-word' queries, these are the individual words, and in the case of structured queries like CQL, these are the terms of the queries.

It is not assumed that the queries and the documents live in the same vector-space, nor that all the documents are following the same document model, or are indexed using the same ontologies.

In the case of LQRTs, the best a new peer can do is to forward a query to all its known neighbors - hopefully soon hitting a peer which has already a well established query routing table, and can do better than broadcasting. It might also choose to send queries which it does not know how to route to a random selection of its known neighbors. The allowed hop count of the forwarded query is decreased to avoid infinite message streams roaming the network.

As soon as a query has been answered, the reply is routed back along the same path, and any peer on the path can update its LQRT according to the atomic parts of the query, and the

information contained in the hit set. One can choose to add the address of the originator of the hit set to the LQRT, or one can choose to update the entry of the nearest peer from which the hit set had been passed only. These varying strategies to progressive LQRT refinement allow the query-trickling approach to be tuned in different ways, and the resulting networks measured for performance to determine optimum values.

The next time a query with a given atomic part has to be routed through a peer that has previous seen this atom, the LQRT shows the best direction to try. Probably there should be some occasional randomness in query forwarding to make sure that yet unknown parts of the network are queried from time to time.

In the case of DQRTs, the global distributed routing table is updated by each originator of a query response (in form of a hit set, that is). Then the hit set may be returned immediately to the querying peer - there is no need to traverse the same path back. The DQRT is implemented analogously as DHTs, using a ring structure, an XOR tree structure, or whatever works.

2.7 Properties of query trickling

Local data The key property of the LQRT model is that all knowledge is local: documents in the repository, how they're indexed, host tables, routing tables, ontologies, etc. The only global structure in DQRTs are the distributed indexes over the atomic parts of the queries.

Index size The DQRT indexes only the queries sucessfully answered, not the entire space of existing documents as in DHTs. This makes the global index structures much more lightweight.

Local policies In the case of LQRTs, individual peers can and should make their own policy about whether to fan queries broadly over short distances or narrowly over long distances; send randomly, or use local routing tables, etc.

User driven Query routing tables are only based on queries issued by users, that is, they are adaptively refined according to the use pattern of the network. In the case of LQRTs, each peer's ability to route queries specializes according to local use patterns.

Query type Works for 'bag of word' queries as well as structured queries. Even different types of structured queries can be supported in the network: the only thing needed in common is consensus on what the atomic parts of queries are.

Legal issues Both DQRTs and LQRTs work for documents that may not be indexed over the network due to IPR or other legal issues.

Any Document There is no assumption about type or structure of documents, nor of mime type, character encoding or binary form. Any type of document repository is allowed to enter the peer network as long as it understands some of the query types in the system.

Network traffic Less network traffic for maintenance of global structures than using DHTs, since the indexing structure on DQRTs are smaller (due to the fact that queries contain less atomic parts than documents, and that DQRTs are built according to usage patterns.).

Mixing It is possible to mix an overlay network of DQRT peers with ordinary LQRT peers in one and the same peer network. Any peer can decide by itself if it wants to be a DQRT or a LQRT peer. A LQRT peer routes a query according to its local knowledge, and eventually it will hit a DQRT peer which makes a global lookup and routes it directly to the right set of peers to answer the query. The answer is then either routed locally (backtracking) or globally (direct contact), according to which types of peers it went through.

Guarantees Neither LQRTs nor DQRTs guarantee that the best document to a query is found and included in the hit set. They give a good document hit set with high probability when the query routing tables stabilize. But a new query might consist of atomic parts that have yet not been indexed properly.

Replication Replication of documents can be allowed on a peer to peer basis. Dead peers which do not allow replication of their content can not be asked to send documents. On the other hand, peers allowing document replication will add failure-resilience to the network.

A Web links

A.1 Information Retrieval Protocols

ZING: http://www.loc.gov/z3950/agency/zing/

SRW: http://www.loc.gov/z3950/agency/zing/srw/

SRW: http://www.loc.gov/z3950/agency/zing/srw/spec-index.html

Z39.50: http://lcweb.loc.gov/z3950/agency/

Z39.50: http://lcweb.loc.gov/z3950/agency/document.html

Z39.50: http://www.niso.org/standards/standard_detail.cfm?std_id=465

A.2 Structured Queries

CQL: http://zing.z3950.org/cql/intro.html

CQL: http://www.loc.gov/z3950/agency/zing/cql/

CQL: http://www.loc.gov/z3950/agency/zing/cql/sample-queries.html

A.3 Metadata

DC: http://dublincore.org/

DC: http://dublincore.org/documents/dcmi-terms/