Distributed Internet Queries

Panayotis Vryonis http://vrypan.net/

9 October 2004.

A proposal for a decentralized search engine system –food for thought.

Intro.

The information that exists on the Internet is vast and more is accumulated everyday. By central indexing, search engines provide order and a certain degree of organization to this chaotic pool of information.

This approach, however has some drawbacks:

- time lag. Search engines rarely know of recently updated information since they index periodically
- ignorance. A search engine is agnostic as far as the underlying structure and quality of the information found on a web site is regarded.
- web-centric. only information exposed trough a web page is indexed.

The introduction of a P2P search standard would allow us to "ask" for information on a (possibly) real time context, use the internal knowledge of each information storage and expand search beyond web pages.

Current situation

Currently, search engines "crawl" web pages and index their contents. "Crawling" consists of starting from a certain set of web pages (usually acquired from a "directory" such as DMoz `<http://dmoz.org/>`), indexing the content of these pages and following the links they contain.

Some time ago search engines would only index the content of HTML and plain text pages, nowadays many of them will index MS Word, PDF and other documents as well. Some of them will even index images, relying on the context, ALT tags and links leading to them.

Indexing words and phrases is usually not enough. When a user searches for a certain term he needs more than all the pages including this term. He or she needs to find the most “important” pages or the most “relevant” pages. Each search engine has an internal way of valuing “importance”, so that more “important” results appear higher than “less important” ones. Google’s “Page Rank” is probably the most well known among these methods of valuation –and an intuitive one as well.

Even though search engines become more and more “smart” they have to do a lot of guessing. They may have to guess things like character sets/language and to identify same pages with different URLs. They also have to disregard “common” words such as articles. Depending on the language, they may even have to identify the same word in many forms (ex. in Greek verbs will change depending on the subject much like in English we say “I am, you are, he is”).

The process of “crawling” the whole Internet takes time, as a result the information indexed is almost always outdated in the sense that it may have changed since the page containing it was last indexed. And since search engines mostly index web pages, information that is not exposed through such a page is hard to find.

Introducing DQS

I think that the next step in Internet searching is an easy to use Distributed Query System, DQS. The idea behind such a scheme is to delegate searching to the ones that know better - the servers/ systems holding the data.

The idea is quite close to P2P file sharing systems:

1. A node is given a query
2. If it knows the answer it returns it (the answer is in this case a set document description information including it’s URL as well as other metadata information such as document title, modification date, etc.)
3. The node may also return URIs to other nodes that could be queried for the same information.
4. Recursively, we may go on asking nodes until satisfied with the results gathered.

The above approach has some good advantages, the main being that each node can have it’s own “intelligence” depending on the data it holds. A news server can ignore common words (like “and”, “or”, “the”) while a dictionary server may respect them and return grammatical information regarding them. Given a 4-digit number, a library directory may return books published at that year, while a telephone directory the area using this number a prefix, and so on.

An other important advantage of the above approach is the selection of “suggested” nodes returned by each node. This is close to Google’s Page Rank system, in the sense that a node knows (or could know) better where to look for related information.

The DQS model has also some obvious disadvantages. These include increased traffic between websites, and additional complexity to the web infrastructure (a server has to answer DQS questions in addition to simple HTTP requests) and administration. Additionally, it is vague how results should be valued in order to distinguish and rank “information quality”.

An other side-effect could be that the “search topology” of the web will be changed. Depending on which node we start from we may get different results for the same search. This is not necessarily a disadvantage but something to be taken into account.

Suggested Implementation

When I originally wrote this paper, I thought that the simplest way to implement DQS whould be to imitate the Google WEB API. The Google Web API does not include any P2P features. However, it provides a nice and simple way to query Google. It is based on SOAP, is easy to implement in almost every programming language, is well documented and easy to extend in order to cover the needs of DQS. The original idea is to implement the Google Web API on websites and extend this API to support the described DQS functionality.

Since then, I have come to thing that it would be much simpler to implement DQS using simple XML POST (over HTTP). I am not an XML expert so I will try to make the examples so simple that will not even suggest a DTD (document type definition). I would rather leave this to experts and wait for suggestions.

However this is how I imagine a search using DQS would work

Initial Request

The initial node is sent a POST request that looks like this:

```
<query>
 <terms>
        <operator value=”and”>
         <item>php</item>
           <item>date functions</item>
            <item>examle</item>
        </operator>
```

```
    </terms>
   <timeframe> ...</timeframe>
    <lang>fr</lang>
</query>
```
(Side-note: I think it would be nice to use a DTD that can be easily transformed to SQL queries.)
Response The node depending on its knowledge will return an XML document containing (possibly) 2 sections: a) “local” results (i.e. results known to this node) and b) a list of suggested URLs to look for more results. One or both of these sections can be empty of course..

```
<results>
    <item>
     <title>Item Title</title>
      <url>http://server/page.html</url>
     <description>This is a description/abstract of the result</
description>
    </item>
    <item>
 ...
    </item>
</results>
<nextnodes>
 <item>
     <title>A server that could Know More</title>
       <url>http://...</url>
      <description>My friend's blog</description>
    </item>
    <item> ... </item>
 ...
</nextnodes>
```
Evaluation

Depending on the results returned, we may stop or go on asking the same query to the nodes described in the `<nextnodes>` section.

Problems to be solved

Keeping track of visited resources

A client should be able to remember that it has already queried a URL and in case an other site suggests so, it should not query it again. Apart from being more efficient, it will make it more difficult to get int endless loops.

Results ranking/ordering

OK. Everything works fine and we end up with 1000 results. How do we order them? Which ones should be at the top?Will we ever get to ask the right site?

Unlike P2P file sharing, we have a much bigger number of possible results. In file sharing (ex. Music files), it is assumed that the same resource can be found in many locations. On a DQS model, we are often looking for a resource that is not duplicated around the net. Depending on the topology created by DQS, and the first node(s) queried, we may be too far away from our target.

Implementing DQS will be complicated for most users.

This is true on “personal” and “small” websites, but it should not be crucial. First of all, an important percentage of these sites use standard engines nowadays, like phpnuke, drupal, wordpress etc. In addition, an increasing percentage of web-content is created using what seems to become the “standard web publishing platform”, blog engines. So, DQS functionality should come with the engine, or even hosting much like webstats. Even if this is not the case, there could be “third party” DQS servers where one can register a website, much like we do with search engines now.

The role of big search engines in a DQS world

So, will big search engines become obsolete if DQS is widely used? I do not think so.

First of all they will become even better since they will have a better, more intelligent way to acquire and evaluate data. So, expect them to give you better results.

Then they could become something like the connecting point between disconnected (DQS-wise) areas. Giving a DQS-enabled site that has computer-related content a tourism-related query, is not likely to return a usable set of results unless it redirects you to a “higher” or “general purpose” DQS server – and a perfect match for this role is a big search engine.

What’s more, search engines like Google, Yahoo! and altavista are more than efficient Internet indexers. They are usable, functional websites. They have invested a lot in usability, speed, localization, synergies. Users need all that.

About this document

I am a freelance programmer, and any knowledge I have regarding the way search engines work and make money is based on things I read around –I have no way of knowing how accurate they are.

I am aware that many of the ideas described in this document will bring acronyms such as RDF, RSS, ATOM, OPML and SOAP to a web developer's mind. I intentionally choose not to refer to specific technologies or standards, as same of them are conflicting and there are others more qualified to decide which one fits better DQS.

In any case, this document should be considered "food for thought". Please let me know what you think!