pSearch

A re-entrant distributed, peer to peer search system.

Evan Stawnyczy
10/27/2011

Contents

Forward 3
Commentary 4
Motivations 4
Location Services 5
Ads 5
Censorship 5
Data Replication 6
Failings 7
Expert Peers 7
Context 8
The Peer Database 8
Syntax 8

Forward

It seemed stupid to me that search giants like Google, yahoo, and Bing replicate the sites that they're indexing in their own database. This replication is expensive, both in storage, and bandwidth/transfers. Even if they only index a small amount (relatively speaking) they STILL cannot produce up-to-date "live" search results.

This waste is becoming more evident especially as the popularity of the dynamic web grows. Twitter and sites that allow constant, up to the minute, content changes will frequently have content that is not indexed until the next indexing interval. In reality Google and likely others have a "backdoor" into the twitter database to increase the accuracy and speed in which they can be indexed, giving an advantage to the larger "twitter" and other large, hard to keep current, sites; whereas Joe the mircoblogger software does not get this same opportunity to compete.

I intend to address these issues.

I am proposing a distributed, controlled and globally equal environment – let everyone now index and serve search requests themselves. This means that, as long as your requesting searches from the "right places" your search results will be against the very latest content of the site, and not against the last time they checked. A "live" search, if you will.

This live searching will also allow searching to be more flexible as well.

Sometimes a keyword, or set of keywords have different meanings, depending on context. This allows sites to do their own searching, giving them control over how keywords are indexed, searched and even evaluated. Words and phrases that are meaningful to each site can be addressed itself, instead of relying on a third party to properly assess or address your search terms.

Commentary

Motivations

The design of this search algorithm is suppose to be expandable, and resistant to invalid data.

pSearch is designed to run using a UDP (user datagram protocol) transport protocol. This is so that it has low computational overhead, and has been designed to suffer a high amount of data loss without problems and can efficiently serve a large number of clients very quickly.

When a node receives a “search request” command, it uses a simple hash to create a hexadecimal identifier that can be used to reference this search request again.

The search node immediately returns the hash value to the client, who can reconnect at anytime to request more data.

By default each node forwards the request to its peer network; this is completely configurable. Peers who are marked as “expert” nodes and have keywords that match or match closely to the search terms set are favoured to anonymous peers and are requested first. Private peers are favoured to anonymous peers, so the search order (also configurable) is: Expert, Private, and then Public by default.

On a search request, the node will immediately search its own “Library” for valid references to return and return those as a start to its cache. If no data is available, no reference will be returned.

Periodically, each node will review its requests and request further information from its peers by issuing a continue command and passing the hashed identifier. Each node is expected to respond with its top results at the time of query. The number of results, normally greater than 10, is configurable as well.

If a node receives a large number of results for a specified set of keywords, the keywords are recorded and the node is marked as an expert.

Once all cycles of this search are complete, the temporary cache of results is destroyed (or cached for library maintenance if configured to do so). Before destroying the peer cache, the top experts (normally 10, but this is also configurable) are posted to the peer database.

Using this set of rules a node who returns a lot of results, quickly, for a specific set of terms becomes ‘favoured’ for these terms.

Peers can request an exchange, however only anonymous peers are going to be exchanged, by default. Peers can be authenticated, through an out of band request, a user can authenticate a specific peer, or set of peers that will be queried on each search response, but not shared with anonymous peers. This will allow for private search networks within a larger network of search nodes. This means that an external system, such as some system within a DMZ could have no Library attached to it (only cache) and it would forward ALL its requests to internal systems. This would create an effective search network distributed within a network.

Location Services

Often you will use something like “google.ca” for Canada, and “google.com” for Google USA, and each site will deliver different results depending on location. This is quite clever of Google and other search providers. While searching in Canada, I may not be interested in what the Iraq version of search results are.

Since the entire system is developed with a giant mesh of users in mind, physical distance starts to make an appearance here. That is, the closer, physically, the peer is, the faster its results will return to my database, and the sooner these results are published to my results cache. This virtually eliminates the need for all these extra “google.ca” and “yahoo.ca” sites. Instead, a peer node that is closer will respond to my request giving me relevant information based on the fact that they simply live closer to me.

Even from an organizational standpoint this is helpful. An organization who wants to have a worldwide presence can place a single node, equipped with a private search network to its “home base” (wherever in the world that may be) in each physical zone, forwarding requests to its “home base” and responding as if it were local. This reduces the cost of having independent datacenters since all search routing is done for it by the internet.

Consider:
I speak French, and I want my results in French. So I do a simple search, since all my peers are nearby me, and I live in a French speaking part of the world. Their databases will be available to me. They will return the French alternative for me since this is what appears in their collection points.

Ads

Yes, this world will also change. Nobody now can pay anyone else to give their site a higher ranking than anyone else. This will level the playing field, so that Joe the microblogging site has equal opportunity at the microblogging market as twitter, both self indexed, and easily searched. This means that quality will become the deciding factor. What other people actually USE will return higher results than whoever has the most money. This freedom is a large factor of why I designed this engine. This functionality allows the USERS to chose which sites become most popular, and not who has the most money or marketing power.

Users can also collaborate and create their own distributed network by authenticating other peers in their own database and using the internet as a transport agent. This configuration allows for private search networks to exist –cooperatively- within a larger set of nodes.

Censorship

Traditional search providers own the data that they distribute. This also allows them to censor the results you receive. In some case (such as Google) they don’t do such a bad job, in others (yahoo for example) do. I believe that we should be free to censor our OWN search results. And I have designed this to do just that. Allow self censorship, without affecting others. See, you can provide a list of censor data which will censor all incoming requests and drop the terms you want censored. This would allow

for you to censor your own database, your own results and the results that you will provide to others, without affecting how another person searches.

Consider:

I don't like Acme Corporation, they wronged me in the past and I do not want to support their business in any way. So, I added the word "Acme" into my censor list. This means that when Billy, who is a peer of mine, requests my database for "Acme products" my database will immediately drop the "Acme" and search only for "products." This may not be what Billy is looking for, but that's what my database will give, probably burying my results below that of the actual "Acme products" website.

Consider Also:

I work for a company called Awesome Co. "Awesome Co" competes with "Lame Co." We both produce a product called "The Product." So I can easily filter out any requests for "Lame Co." And only reply to "The Product." Or, inversely, just not respond to any queries that have "Lame Co." and "The Product" in the keywords.

Data Replication

Yes, in my forward I mentioned that there is little, to no indexing. This is true, but misleading.

Private clients, such as home users would have their own small database indexing sites "of interest to them" Interest is determined from a variety of collection points, which, I've tried to develop as modular as possible. This allows you to choose which collection points would be useful, or use none at all. I aimed to include DNS, and browser caches as good initial collection points, however scanning results delivered by its peers also produces some excellent collection points as well.

Of course a client will also forward all search requests to its peers to obtain better results.

Consider:

My wife constantly likes to check twitter for the latest and greatest quotes from her favourite celebrities. One day, she's telling me about something that someone said, but cannot recall who, or when they said it. So, she enters a couple of words from what she remembers into the search, and because "twitter" appears in her browser history so frequently, it's been indexed by her local database. The quote she was searching for comes from her OWN MACHINE, without need to go any further it shows up immediately, no wasted bandwidth and quite often, faster results.

Service providers, such as websites and hosts like that can link into their own back-end databases so that their search results can accurately provide full LIVE results from their own site allowing instant indexing of their published content for the public to search at their leisure. This also allows providers the ability to redirect client to proper locations where their search results would be most useful.

Consider:

I have an EvansBrand computer model M100. I need a graphics driver for its crappy built-in EvanGFX 100 card. So, I do a search for "EvansBrand M100 EvanGFX100 driver" or similar. This request ends up at my site thorough the web of peers and I realise it's talking directly about my brand and my driver. I know that the driver page is at "/evansbrand/m100/drivers/graphics" but I also know that whoever needs this driver will possibly be interested in the other up to date drivers that are published on my site. So, instead of returning "/evansbrand/m100/drivers/graphics" as the path, I can return instead: "/evansbrand/m100" the summary page where I'd want them to land anyway.

This is the power of an independently controlled live search.

Failings

This control also produces some problems. What is to stop someone from inserting their own site as the answer for ALL results?

Is this is truly a problem?

NO!! As search results come in from peers, repeated results are sorted to the top of the database. As if someone else endorses this site as well. Therefore, as long as more peers return correct results, the correct results will be displayed.

Expert Peers

By default, each peer will try to provide the "best" results to its clients. However, how does a peer decide what are the "best" results?

Peers can judge results by how frequently they appear from other peers. This effectively allows for a separate peer to "endorse" this result. Endorsements are only accepted on the first response, so that, a peers private database can be "endorsed" and not a peers peer. (Confusing?)

A rapid response with multiple references will get a peer marked as an "expert." An "expert" peer will be queried again when similar keywords are queried.

It is preferred that service providers, such as web sites do not return results that are not relevant to their own site, in order to quickly make all peers find them "experts". Therefore, when someone searches for specific keywords, or sets of keywords, the service providers' site can respond with accurate results itself, and quickly be endorsed by other peers.

Consider:

Some user queries my engine for "IBM." I may have a reference or two about IBM, but the majority of my response is going to my peer network. Now, as it happens, I have ibm.com as one of my peers. Since the keyword "IBM" matches almost every page on their site, the ibm.com peer will reply with more than enough results for my engine. My engine will in-turn mark ibm.com as an "expert" when the keywords contain the word "IBM." Next time my engine gets a query for "IBM," it will automatically forward the request to ibm.com, the "Expert" and have more results, quicker.

Context

The internet is quite vast, and is only growing.

There are literally hundreds of file types, each with their own strengths and weaknesses. Each file type has its own metadata and method of storing additional information. This is why I focused on file reading or "parsing" engines as well. This allows the search engine to properly index each file type independently. This allows the same indexing engine to index mp3 files by reading the entire mp3 metadata stored in the id3 tag, and allowing searches against the metadata. This is relevant for all file types that contain more data than just its filename.

Since this search service is delivered outside of the HTTP protocol, it's not limited to HTTP. This means that it can index all URL types, including FTP, SFTP and even newsgroups. It can be used to index physical mediums such as books and extensible enough to index data that changes down to the second.

Consider:

I wrote a great piano song and I want everyone to hear it. I upload it to my favourite mp3 sharing site and hope that everyone listens. Someone else is interested in my song, so I tell them, search for Artist: evan and look for the 128kbps encoding. Just entering this information into Google will not bring up my mp3, unless it's indexed in their site (unlikely since Google only indexes pictures or web content) instead however since the indexing is done by a process that can easily read id3 tags, all the data in the id3 tag will be easily represented and indexed... by the same engine that is indexing web sites. Now consider this applied to archives (search within archives?) how about bittorrents (search within bittorrents) or even within a single ISO (why not?)

The Peer Database

The peer database is the heart of where the search engine will find its results. A quality group of peers can provide very in-depth search abilities.

The peer exchange process allows public peers to gather other public peers to help themselves create their own expert peer lists based on searches they've received. By default, each peer will share up to 10 (yes, configurable) randomly selected public peers per peer request cycle. Private peers can only be added manually.

Syntax

I wanted pSearch to use a simple ASCII based protocol, such as HTTP. It also is designed to run on UDP, and therefore is built with the idea that each request will reply once. So, each request will be given a reply, the connection will be closed, and a new request will be created for any further queries.

A simple keyword search:

>KEYWORD testing

<IDENTITY ffeeffee00

<REFERENCE

<TITLE: Testing

<URL: www.testing.org

<REFERENCE

<TITLE: Testing second site

<URL: www.also-testing.org

>STATUS ffeeffee00

<REFERENCES: 100

<PEERS: 10 /1

<EXPIRES: 4:30 PM (GMT)

>STATUS

<VERSION: 1.0.0 alpha (Anxious Longprince)

<URLS: 1024

The STATUS Command

The “Status” command will return the status of a query, or the server. To query the status of a server, a simple STATUS command would be issued. The server should reply with at least the version of the protocol supported, additional statistical information can be configured, but is not required.

When the “Status” command is issued against a query will return, at least, the time and date the query will be expired. Additional information can be configured, but is not required.

The KEYWORD Command