IN-2194 Project: Search

1 General Information

The goal of this project is to implement a Peer-to-Peer Search Engine. You should read up on `http://yacy.net/` for an existing design.

Unlike traditional search engines, your engine should assume that it is given a list of domains to index. Crawling should then be limited to those domains only.

The suggested high-level approach is that you begin by issuing a "DHT PUT" operation for each domain to index. The peers that receive those DHT PUTs (using the DHT monitoring API) are then responsible for crawling the respective site. As the crawlers download the site, an index must be created. This will depend on how the search is done later; a simplistic method would be to do "DHT PUT" operations (using a different type) on each keyword found. Searching the DHT would then only require "DHT GET" operations on the individual keywords. Alternatively, a local index might be created with the search then operating using flooding. Other styles might also work, and the choice is yours.

You must submit your solutions using Subversion by **August 15th 2012**. Late submissions will receive a score of zero unless you notify the instructor about a valid reason (such as the entire group having been abducted by pirates) **prior** to the deadline.

1.1 Subversion Access

In order to receive Subversion access, you must submit a request at

`https://projects.net.in.tum.de/projects/tum2194/register`.

In this request you must include the login name and desired password. Your login name should be your lastname followed by the last two digits of your student identification number.

Once your account(s) have been created and teams have been assigned, you can access the repository at

`https://projects.net.in.tum.de/svn-ext/tum2194/TEAMNAME/`.

You must submit a request for account and team creation by **Mai 15th 2012**. We want you to use Subversion for development, not just submission! Note that only the version of your code that is HEAD in the repository on August 15th 2012 at 23:59 will be graded.

1.2 Coding Style

You should read the GNU coding standards for a reasonable guideline for writing good C code. In general, a shorter and simpler implementation will receive higher marks than a complex and long implementation.

Naturally, comments and testcases will not be counted against you when we evalute the size of the code (in fact, good testcases and comments may earn you style points).

Your implementation must not perform busy-waiting and must not use pthreads (!). You should also try to avoid (unnecessary) copying of data. You should stick to the GNUnet coding style as much as possible (if your implementation would be good enough to be merged without significant changes into the GNUnet code base itself, you will get full points).

2 The GNUnet Framework

Your implementation should work in the context of the GNUnet P2P framework. Specifically, you will be using the 0.9.x development branch which can be found at `https://gnunet.org/svn/gnunet/`. Some documentation is available online at `https://gnunet.org/`. The framework already contains a DHT, including a C API `src/include/gnunet_dht_service.h` that is used by other components to access the DHT and command line tools (`gnunet-dht-put` and `gnunet-dht-get`).

In GNUnet, each component is executed as a separate process. Your implementation will use the *core* service and/or the DHT to communicate with other peers. You can use the *peerinfo* service to bootstrap (discover peers initially).

Each of the services listed above provides a service access library (`libgnunetSERVICE.so`) to communicate with the service process that actually performs the operations. For this, the service process listens on a socket for requests from the service access library. Service processes are managed by the *arm* service.

For your own services, you need to write a service access library. For your implementation, you can choose to write the service process in C/C++ or Java. However, the Java API for some of the access libraries (to *core*, *dht*, *statistics* and others) is still under heavy development; thus the recommended development language is C/C++. If you choose to write your code in Java, you may have to deal with (more) bugs or other limitations in the existing access libraries.

Finally, you can ask questions not answered in the FAQ on the webpage in the `#gnunet` IRC channel on `irc.freenode.net`. We will answer non-FAQ questions in the public channel quickly. We will not answer questions about how to use GNUnet asked elsewhere.

3 Challenges

- Keyword normalization / spell checking: if you are using the GNUnet DHT to store search terms, it will be difficult to do approximate searches. Using a different DHT, normalizing keywords or a different search strategy might help here.

- Search results from a distributed crawl are likely to arrive incrementally, and possibly the ranking of a result may change after it is initially obtained. A nice user-interface would need to dynamically update

the page without ever changing the URL that the user is actively trying to visit.

4 Grading

You are expected to form teams of two to three students, but you can also work alone. Teams of four and more students are not permitted. Within each team, you are free to determine which team member is best for which task. Your overall grade will be determined primarily by the overall quality of the deliverable, but we will also discuss your contributions with you during the final demonstration and make sure that individual students are not penalized if some team members failed to contribute.

For the first three subprojects, the grading will be based on the following main points:

4 Domain-name injection command-line tool

10 HTTP crawler (libcurl, libtidy)

12 Keyword search and user-interface

10 Interesting solutions to some of the hard problems

6 Testcases with good code coverage

4 System documentation

4 Correctness of the implementation with respect to the documented design