Team Project Specifications

CIS 455 / 555 – Internet and Web Systems

Spring 2009

Zachary Ives

"Code Complete" Deadline: May 5, 2009
Final Report Deadline: May 8, 2009, 6PM

For the term project, you will be building a peer-to-peer Web indexer/crawler and analyzing it with respect to its performance. This will involve several components, each of which is loosely coupled with the others:

- Crawler
- Indexer/TF-IDF Retrieval Engine
- PageRank
- Search Engine and User Interface
- Experimental Analysis and Final Report

More details on each component are provided below. The project is relatively open-ended and includes many possibilities for extra credit. However, you are strongly encouraged to get the basic functionality working first.

Suggested approach: Spend some time early coordinating with your group-mates and deciding which modules from your previous homework assignments are "best of breed." Designate one person to be responsible for each task.

Make sure adequate time is spent defining interfaces between components (in this case, appropriate interfaces might be Pastry messages, Web service calls, and perhaps common index structures), and also plan to spend significant time integrating. Consider the use of automated tools for building your project (ant) and version control (cvs or svn; see http://www.seas.upenn.edu/cets/answers/subversion.html).

Note that the report includes a non-trivial evaluation component.

You should make use of (1) the mod cluster for concurrent execution, (2) subversion for sharing your work, (3) JUnit tests for validating that the code works (or remains working).

Crawler

The Web crawler should build upon your past homework assignments, and it should be able to parse typical HTML documents. It should check for and respect the restrictions in **robots.txt** and be well-behaved in terms of concurrently requesting at most one document per hostname. Requests should be distributed, Mercator-style, across multiple

crawling peers built over Pastry. The crawler should track visited pages and not index a page more than once.

Extra credit: Add support for digests to detect when the same document has been visited more than once. If so, the document should only be stored once – but two "hits" should be returned.

Indexer

The indexer should take words and other information from the crawler and create a **lexicon**, **inverted index**, and any other necessary structures for returning weighted answers making use of TF/IDF, proximity, and any other ranking features that are appropriate. It should be able to store data persistently across multiple nodes, using Pastry and BerkeleyDB.

Extra credit: Include document and word metadata that might be useful in creating improved rankings (e.g., the context of the words).

Extra credit: Make the indexing system restartable with a different number of peers. The *simplest* way to do this is to have a procedure where, at startup, each node with an existing BerkeleyDB index will (1) rename it, (2) create a new index file, and (3) scan through each entry in the old index file and PUT it into the DHT.

PageRank

Given information from crawling, you should perform **link analysis** using the PageRank algorithm, as discussed in class and in the Google PageRank paper. Your implementation does not need to be distributed.

Extra credit: Implement a distributed version of PageRank, e.g., following one of the approaches of the following papers:

http://www.cs.utexas.edu/users/simha/publications/distributedpagerank.pdf
http://wwwcsif.cs.ucdavis.edu/~yeshao/cikm05.pdf

Search Engine and User Interface

This component is fairly self-explanatory, as the goal is to provide a search form and results list. One aspect that will take some experimentation is determining how much to weight each item (PageRank, TF/IDF, other word features).

Extra credit: Integrate Yahoo search results into your keyword listings, using the REST interfaces. A challenge here is how to interleave ranked results.

Extra credit: Integrate Amazon search results into your keyword listings, using the REST or SOAP interfaces. A challenge here is how to interleave ranked results, especially given that some topics may be more or less suited to Amazon.

Extra credit: Implement a simple Google-style spell-check: for words with few hits, try simple edits to the word (e.g., adding, removing, transposing characters) and see if a much more popular word is "nearby."

Extra credit: Consider adding AJAX (Asynchronous Javascript And XML) support to your search interface, so users can provide feedback about which entries are "good" or "bad," and use these to re-rank the results.

Experimental Analysis and Final Report

Building a Web system is clearly a very important and challenging task, but equally important is being able to convince others (your managers, instructors, peers) that you succeeded. We would like you to actually *evaluate* the performance of your methods, for instance relative to scalability.

One approach is to write a simple query generating tool that poses many queries at once, and compare response time with one, two, up to *n* peers (where *n* is the maximum number of machines in the mod cluster). What is the maximum number of concurrent requests you can reasonably handle, when varying the number of threads in a thread pool and the number of nodes? Can you separate out the overhead of the different components (including network traffic)?

Your final report should include at least:

- Introduction: project goals, high-level approach, and division of labor
- Project architecture
- Implementation: non-trivial details
- Evaluation
- Conclusions

Note that the quality of the report *will* have substantial bearing on your grade: it is not simply something to be cobbled together at the last second!