CS6200 Information Retrieval

Homework3 : Crawling, Vertical Search

Objective

In this assignment, you will work with a team to create a vertical search engine using elasticsearch. Please read these instructions carefully: although you are working with teammates, you will be graded individually for most of the assignment.

You will write a web crawler, and crawl Internet documents to construct a document collection focused on a particular topic. Your crawler must conform strictly to a particular politeness policy. Once the documents are crawled, you will pool them together.

Form a team of three students with your classmates. Your team will be assigned a single query with few associated seed URLs. You will each crawl web pages starting from a different seed URL. When you have each collected your individual documents, you will pool them together, index them and implement search.

Although you are working in a team, you are each responsible for developing your own crawlers individually, and for crawling from your own seeds for your team's assigned topic.

Obtaining a topic

Form a team of three students with your classmates. If you have trouble finding teammates, please contact the TAs right away to be placed in a team.

Once your team has been formed, have each team member create a file in Dropbox named team*XY*abcd.txt (using your first initial and the last name). This file should contain the names team members. The TAs will update this file with a topic and three seed URLs.

Each individual on your team will crawl using three seed URLs: one of the URLs provided to the team, and at least two additional seed URLs you devise on your own. In total, the members of your team will crawl from at least nine seed URLs.

Crawling Documents

Each individual is responsible for writing their own crawler, and crawling from their own seed URLs.

Set up Elastic Search with your teammates to have the same cluster name and the same index name.

Your crawler will manage a *frontier* of URLs to be crawled. The frontier will initially contain just your seed URLs. URLs will be added to the frontier as you crawl, by finding the links on the web pages you crawl.

1. You should crawl at least 20,000 documents individually, including your seed URLs. This will take several hours, so think carefully about how to adequately test your program without running it to completion in each debugging cycle.
2. You should choose the next URL to crawl from your frontier using a best-first strategy. See Frontier Management, below.
3. Your crawler must strictly conform to the politeness policy detailed in the section below. You will be consuming resources owned by the web sites you crawl, and many of them are actively looking for misbehaving crawlers to permanently block. Please be considerate of the resources you consume.
4. You should only crawl HTML documents. It is up to you to devise a way to ensure this. However, do not reject documents simply because their URLs don't end in .html or .htm.
5. You should find all outgoing links on the pages you crawl, canonicalize them, and add them to your frontier if they are new. See the Document Processing and URL Canonicalization sections below for a discussion.
6. For each page you crawl, you should store the following filed with ElasticSearch : an id, the URL, the HTTP headers, the page contents cleaned (with term positions), the raw html, and a list of all in-links (known) and out-links for the page.

Once your crawl is done, you should get together with your teammates and figure out how to merge the indexes. With proper ids, the ElasticSearch will do the merging itself, you still have to manage the link graph.

Politeness Policy

Your crawler must strictly observe this politeness policy at all times, including during development and testing. Violating these policies can harm to the web sites you crawl, and cause the web site administrators to block the IP address from which you are crawling.

1. Make no more than one HTTP request per second from any given domain. You may crawl multiple pages from different domains at the same time, but be prepared to convince the TAs that your crawler obeys this rule. The simplest approach is to make one request at a time and have your program sleep between requests. The one exception to this rule is that if you make a HEAD request for a URL, you may then make a GET request for the same URL without waiting.
2. Before you crawl the first page from a given domain, fetch its robots.txt file and make sure your crawler strictly obeys the file. You should use a third party library to parse the file and tell you which URLs are OK to crawl.

Frontier Management

The *frontier* is the data structure you use to store pages you need to crawl. For each page, the frontier should store the canonicalized page URL and the in-link count to the page from other pages you have already crawled. When selecting the next page to crawl, you should choose the next page in the following order:

1. Seed URLs should always be crawled first.
2. Must use BFS as the baseline graph traversal (variations and optimizations allowed)
3. Prefer pages with higher in-link counts.

4. If multiple pages have maximal in-link counts, choose the option which has been in the queue the longest.

If the next page in the frontier is at a domain you have recently crawled a page from and you do not wish to wait, then you should crawl the next page from a different domain instead.

URL Canonicalization

Many URLs can refer to the same web resource. In order to ensure that you crawl 20,000 distinct web sites, you should apply the following canonicalization rules to all URLs you encounter.

1. Convert the scheme and host to lower case: `HTTP://www.Example.com/SomeFile.html` → `http://www.example.com/SomeFile.html`
2. Remove port 80 from http URLs, and port 443 from HTTPS URLs: `http://www.example.com:80` → `http://www.example.com`
3. Make relative URLs absolute: if you crawl `http://www.example.com/a/b.html` and find the URL `../c.html`, it should canonicalize to `http://www.example.com/c.html`.
4. Remove the fragment, which begins with `#`: `http://www.example.com/a.html#anything` → `http://www.example.com/a.html`
5. Remove duplicate slashes: `http://www.example.com//a.html` → `http://www.example.com/a.html`

You may add additional canonicalization rules to improve performance, if you wish to do so.

Document Processing

Once you have downloaded a web page, you will need to parse it to update the frontier and save its contents. You should parse it using a third party library. We suggest jsoup for Java, and Beautiful Soup for Python. You will need to do the following:

1. Extract all links in `<a>` tags. Canonicalize the URL, add it to the frontier if it has not been crawled (or increment the in-link count if the URL is already in the frontier), and record it as an out-link in the link graph file.
2. Extract the document text, stripped of all HTML formatting, JavaScript, CSS, and so on. Write the document text to a file in the same format as the AP89 corpus, as described below. Use the canonical URL as the DOCNO. If the page has a `<title>` tag, store its contents in a `<HEAD>` element in the file. This will allow you to use your existing indexing code from HW1 to index these documents.
3. Store the entire HTTP response separately, as described below.

Link Graph

You should also write a link graph reporting all out-links from each URL you crawl, all the inlinks you have encountered (obviously there will be inlinks on the web that you dont discover). This will be used in a future assignment to calculate PageRank for your collection.

- option 1 : We prefer that you store the canonical links as two fields “inlinks” and “outlinks” in ElasticSearch, for each document. You will have to manage these fields appropriately, such that when you are done, your team has correct links for all document crawled.
- option 2: maintain a separate links file (you can do this even if you also do option1). Each line of this file contains a tab-separated list of canonical URLs. The first URL is a document you crawled, and the remaining URLs are out-links from the document. When all team members are finished with their crawls, you should merge your link graphs. Only submit one file, containing this merged graph. During the merge process, reduce any URL which was not crawled to just a domain.

Merging team indexes

Ideally we would like to have the crawling process send any stored data directly to the team-index, while merging. But this is too much of a headache for students to keep their ES servers connected while crawling; so we allow for individual crawls, then merged in ES. If you use individual crawls to be merged at the end, you have to simulate a realistic environment: merge indexes (or the crawled data) into one ES index. Merging should happen as independent agents : everyone updates the index independently while ES servers are connected. Meaning not in a Master-Slave or Server-Client manner. This is team work.

Once all team members are finished with their crawls, you will combine the documents to create a vertical search engine. It is required that team computer/ES are connected are the time of merging, and that each team member runs merging code against the merged index in an independent manner (no master-slave design)

Vertical Search

Add all 90,000 documents to an elasticsearch index, using the canonical URL as the document ID for de-duplication, and create a simple HTML page which runs queries against your elasticsearch index. You may either write your own interface, or use an existing tool such as Calaca (https://github.com/romansanchez/Calaca) or FacetView (https://github.com/okfn/facetview). Or modify this one (../matt_es_client.zip) written by one of our grad students. Your search engine should allow users to enter text queries, and display elasticsearch results to those queries from your index. The result list should contain at minimum the URL to the page you crawled.

Make sure you run several queries on your group’s topic, and you think about the result quality. During your demo, you will be asked to explain how your seeds and crawls affected the search results.

Extra Credit

These extra problems are provided for students who wish to dig deeper into this project. Extra credit is meant to be significantly harder and more open-ended than the standard problems. We strongly recommend completing all of the above before attempting any of these problems.

Points will be awarded based on the difficulty of the solution you attempt and how far you get. You will receive no credit unless your solution is “at least half right,” as determined by the graders.

EC1: Crawl more documents

Expand your team crawl to 100,000 documents

EC2: Crawl directly into a merged index

Instead of individual crawl in advance, crawl and store documents in a distributed ES index, so merging happens dynamically. This requires consideration and a laborious setup, but it is the most realistic scenario:

- have your team computers ES in a stable connection for the duration of the crawl (might take hours!)
- create a distributed index, make sure it spans all computers
- each team member starts the crawl at the same time, and each have to independently verify the status of the crawled documents against the distributed index before making changes
- finish the crawl with several tens of thousands docs in it
- make sure to create replicas of the team index before disconnecting ES serves from each other

EC3: Frontier Management

Experiment with different techniques for selecting URLs to crawl from your frontier. See the Coverage slides for the Seattle section for some suggestions. Does the selection policy appear to impact the quality of pages crawled?

EC4: Speed Improvements

Without violating the politeness policy, find ways to optimize your crawler. How fast can you get it to run? Do your optimizations change the set of pages you crawl?

EC5: Search Interface Improvements

Improve meaningfully on your search engine interface. This may include one or more of the following (or your own ideas). Instead of just showing URLs, show text snippets containing the query terms from the document. Change the visual layout or user interface to make the search engine easier to use, or to make it easier to find what you're looking for. Add domain-specific search operators, or other custom search operators.

Rubric

10 points
You strictly follow the politeness policy
10 points
You chose reasonable seeds, and understand the impact of seeds on the crawl
20 points
You crawl pages in the correct order
10 points
You correctly canonicalize URLs
10 points
Correctly index with ES
10 points
Merge crawled pages with your teammates
20 points
Your group's vertical search engine works correctly
10 points
You can explain the quality of your search results