Yahoogle

Andrew Gaber, Cassandra Sauer, Chris Sarachilli, Davis Knaub, Cindy Nguyen

Enhanced/Added Features

1. Domain Score
2. Word Position Score
3. Anchor Word Score
4. Title Word Score
5. Caching for Indexing and Query Processing
6. Pagination
7. Zip Code Search Returns Map Result
8. Multithreading to Accessing Index Table in Parallel

For our final rank, we incorporated the weights of domain score, word position score, title word score, and anchor word score to establish a more robust ranking for our search engine. Our TF-IDF score, allocated a 30% weight, assesses term relevance within a page's content relative to other URLs. PageRank is weighted at 20%, reflecting the authority and credibility of inbound links. We decided on this distribution because we believe the relevance of the term in the page's content should be emphasized in a search engine.

We assigned a 10% weight to the domain score to reflect the term's relevance within the domain name, although this is weighted less to prevent favoring URLs merely due to their domain match.The domain score consists of gaining points for exactly matching or containing the term in the domain's base url, and the domain's path containing the term.

The word position score, also at 10%, prioritizes URLs where the term appears earlier on the page. The weight of the word position score is also low to mitigate the impact of keyword stuffing on search results. The scoring gives higher scores to terms found in the first 500 words than those appearing outside them.

The anchor word score, weighted at 10%, evaluates the term's relevance from the perspective of external link descriptions. Given the potential for link farms, the weight of this score is low. If the term is found in the anchor words it gains a point.

The title word score, which we have weighted more significantly at 20%, highlights the term's presence in the title tag as a strong indicator of page relevancy. The title word scoring is the same as the anchor word scoring.

One enhancement we introduced was incorporating a URL's features in the index itself. The features for a specific URL such as TF-IDF score, word positions, page title, title words

indicator, and external anchor words indicator were appended to the URL before creating the inverted index with Flame. With these features in the index we were able to avoid creating and accessing additional persistent tables during a query search. Retrieving, scoring, and ranking URLs became a more efficient operation.

Another enhancement we introduced is caching for both indexed results of prior searches and previously retrieved index table rows. This significantly reduces retrieval times, optimizing overall search efficiency.

Additionally, the search engine accesses the index table in parallel using one thread for each word queried.

Finally, we incorporated pagination as a quality of life improvement on the frontend, and searching a zip code will deliver a link to a map of the area as the first result.

Challenges

1. Local Environment Variability: Our team faced challenges with diverse local setups, which affected consistent program execution. We addressed this by creating a README file with instructions and to accommodate both macOS and Windows, ensuring uniformity in our testing environments.
2. Debugging Complexity: The vast scope of our program led to numerous potential failure points, especially within the complex structures of the crawler and indexer. Our team adopted a highly collaborative approach, sharing responsibilities and supporting one another in troubleshooting by assisting each other beyond our specific tasks. This was pivotal in overcoming these challenges and resolving issues collectively.

3 Most Difficult Challenges

1. Scaling Crawler and Indexer: Scaling the crawler posed significant challenges, from managing failures during runs to real web page crawling, optimizing memory usage, and increasing operational speed. We refined our URL extraction and parsing methods, added more rigorous filtering for page extensions, and adjusted network connections to resolve memory leaks to significantly improve our crawler. We finally were able to crawl a reasonably large corpus (around 400,000 pages), we encountered further scalability issues when indexing. Malformed urls or pages that snuck past the crawler suddenly crashed the indexer, and we also encountered memory issues when calculating the inverted indices.

2. Indexer: Developing the indexer for the search engine involved detailed calculations for TF and IDF values necessary for the TF-IDF scores to use in the final rank. We ran into issues with the implementation to grab the right values needed for the calculations but were able to refactor our first implementation once we understood what values we needed

and where they were. Then we had to find the best way to incorporate data in our index table to enhance our final rank. We decided to map each term to have a single entry of comma-separated strings of data. Each data string had the following information included in this order: the url, TF-IDF score, Word Position Score, the title or NULL if there was no title, YES or NO whether the url has the term in the title, and YES or NO whether the url has the term in the anchor tag. We encountered parsing errors due to initially using colons as delimiters, which are common in URLs. We also faced integration challenges with foldByKey operations. By switching to spaces for delimiters and changing the foldByKeyElement to 0, we resolved these issues, enhancing the indexer's reliability and integration with other components.

3. Code Integration and Refinement: Our varying implementation approaches initially slowed our progress. We first decided to select the best performing pieces of code for each grouping of homeworks. However, none of us did exceptionally well on the grouping for the web server so we took the top individual implementations from the group and attempted to merge them. There was a major delay for us to get a minimal solution working for our search engine as it was hard to integrate the code for the web server due to the nature of the various implementation strategies that each person took for each part. The integration process made a breakthrough when we adopted a superior solution from one team member and made the necessary adjustments to function effectively.

Things We Would Do Differently:

1. Initial System Design & Pipeline Framework: Had we established a system design and pipeline framework at the start, we could have had a better visualization of the interdependencies of the project components to reduce confusion and align our development efforts and goals. It would’ve provided clear guidelines and expectations for a more efficient parallel development of the project components.

2. Time Management and Integration: We recognized the need for better time management and learned the importance of integrating our components into a functional whole earlier in the project timeline to facilitate timely identification and resolution of integration issues. This would have allowed us to make better improvements and less last-minute debugging, leading to a smoother and more refined project outcome.