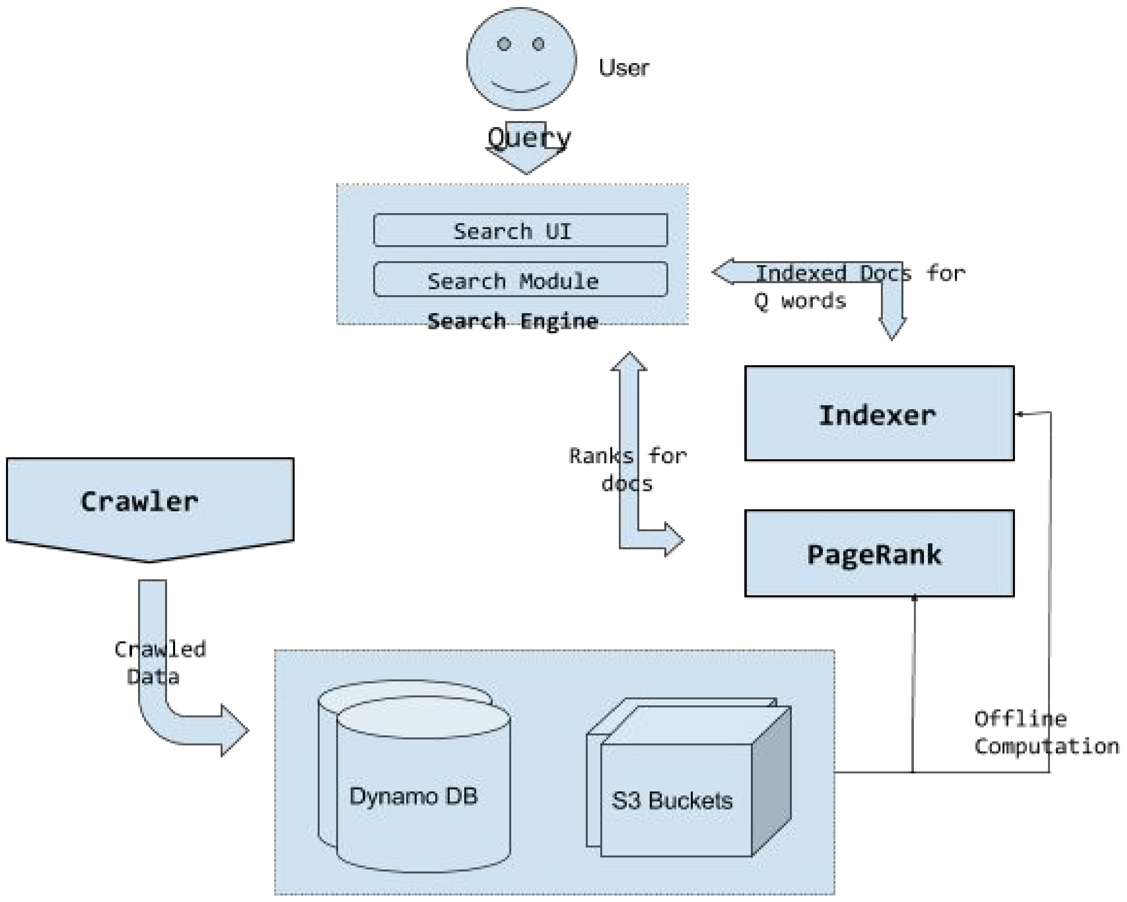
IWS Project Report

Dominik, Gouthaman, Ishan, Shreejit

Introduction:

Our project goal is to design a small, but fully functional implementation of a search engine, **"searchi"**, that runs in the Amazon Web Services cloud. The high-level design is as follows:



To answer a user's search query, the user communicates with the search engine component, from which it sends his query. The search engine first "purifies" the query linguistically before sending it as a list of keywords to the indexer. Based on the list of keywords, the indexer determines all candidate documents that match any of the keywords as well as the document's TF-IDF based similarity score to the search query. The TF-IDF ranked candidate documents are then send back to the search engine, which will refine the indexer's initial ranking proposal by also querying the PageRank component for page authoritativeness. Combining the indexer's and PageRank's ranking and other meta information, the search engine finally displays the ranked query results back to the user. The indexer and PageRank components both serve the search engine's requests based on precomputed tf-idf values and pagerank scores, respectively, that are derived from the crawled raw pages.

Milestones:

1. November, 25th: First working prototype of all 4 components on small dataset.
2. November, 29th: Second integrated prototype on larger data set.
3. December, 5th: Third, expanded prototype on final data set.
4. December, 10th: Final submission.

Github repository tracking progress: https://github.com/bollmann/searchi.

Project Architecture:

Crawler (Owner: Shreejit). The crawler is the input to the search system and supplies url content and metadata to the system. We have chosen to implement it in the multi-nodal mercator style, with a Master allocator running on a single node, and multiple multi-threaded work nodes that will join the Master node and process URLs. The Master node will start processing of the seed urls, and will have a list of nodes where each node represents a domain with associated meta information about crawling pages within the domain. A producer thread on the Master will go through the domain nodes and check if node is allowed to query the domain. If so, it “dequeues” the url and posts it to the job queue (which will be equal in number to the number of worker nodes in the system). The worker then will process the url, store the raw content and associated metadata like size, content type, last modified etc. in the appropriate storage system, and then extract the outgoing links from the page and feed it back to the Master.

PageRank (Owner: Ishan). The overall pagerank system consists of a module that does an offline computation of page ranks and servlet based interface to service the requests for search engine module.

1. Offline Computation Module is essentially a 3-phase Map Reduce job.
    a. Phase 1- Map Reduce job reads link network and puts it in proper format.
    b. Phase 2- **Iterative** Map Reduce algorithm to compute page ranks.
    c. Phase 3- Map Reduce job to aggregate page ranks from link network info.
2. Interface would accept Http requests with a list of page links and the response would consist of the pre-computed pageranks. The pagerank map is going to be loaded in memory initially during startup.

Indexer (Owner: Dominik). The indexer consists of an offline and an online component. The offline component precomputes the inverted index as a Hadoop mapreduce job. As input it takes the raw pages crawled by the crawler, tokenizes them into keywords, and then creates a forward index. The forward index is finally used to create the inverted index as well as the TF-IDF weights for each (keyword, document) pair. The output inverted index is stored in DynamoDB for easy and fast access. The indexer’s online component is built as a servlet-based interface that queries the precomputed inverted index based on the search engine’s HTTP requests. Upon an incoming search query, it looks up all the candidate documents matching the query’s keywords and sends these documents and their TF-IDF scores back to the search engine.

Search Engine (Owner: Gouthaman). The search engine is a Node.JS webapp backed with a Bootstrap framework. The search page accepts a query from the user, and the app performs linguistic cleanup on the search string (such as stop word removal, NLP to determine the keywords in the query) and calls the Indexer with the relevant keywords. In the case of image search, it passes the search string as tags of the image. With the document set returned by the Indexer, it calls upon the PageRank module to obtain the ranking of the documents. Using a combination of both the PageRank and TF-IDF scores, the top ranked results are displayed to the user. AJAX calls are used to dynamically load relevant snippets of each URL, and also to provide user feedback of the ranking which is stored for subsequent use.