Team G22 Final Project Plan 4.10.2016 clancyt, gyixuan, xishanni, adducci

Overview

As defined in the final project overview, this search engine project is the integration of work done on five tasks: the crawler to search for data, TF-IDF-weighted indexer retrieval, MapReduce PageRank result suggestions, the user interface, and a final analysis report.

Responsibilities

Each task will initially have one group member assigned to it. Milestones reflect when various parts of tasks must be accomplished to keep group development moving smoothly. To aid in meeting milestones, group members who complete their assigned tasks or impending milestones will aid those who have not.

Crawler: Rachel will handle the implementation of this component.

Indexer: Yixuan will handle the implementation of this component.

PageRank: Tim will handle this component and then assist with the indexer.

User Interface: Shanni will handle implementing this component. This task relies on the completion of all others, and as such help will be given to crawler implementation first.

Final Analysis: milestones are structured such that all members will have time to dedicate to this.

Development

Development of each component will be done using the VM to ensure compatibility. The group-issued git repo will ensure version control. No extra credit is budgeted into this development plan until all components are verified as working. Our storage solutions will be met by S3. Given that we have such a strong platform to work off of, group members are expected to have their code unit tested and debugged to the best of their abilities by their milestones. There are two special integration milestones for debugging the assembled program in all parts.

Crawler: development of the crawler finds a strong foundation in our HW2 submission, which already meets the requirement of parsing HTML documents, caching and respecting robots.txt, concurrently requesting at most one document per hostname, tracking visited pages, not indexing a page more than once, and identifying itself as 'cis455crawler' in the User-Agent header.

What must still be developed for the crawler is a distributed implementation similar to that described in the Mercator paper and based on the HW3 framework. The “fetch” and “process” stages become implemented in a way similar to our “map” and “reduce” stages, with custom contexts being used to interface with a shared database. Importantly, the new parallelism means that our queue for storing URLs that must be searched may grow larger than any one machine’s memory, and must therefore be implemented using disk. This queue must also consider new politeness challenges, all further detailed in the Mercator paper. For example: the crawler hashes each working node to a particular host. This prevents multiple workers from downloading from the same host and thereby creating duplicate entries. The DataStore interface used by the crawler is abstract enough that it can easily be adjusted away from Berkeley DB. The crawler is closely-tied to the indexer, and must rely on the indexer for all processing.

Milestones: 4/13: ensure crawler has the ability to stop and restart while executing. This improves testing. Obtain and store crawled records for rest of group to test with; 4/17: implement parallelism according to Mercator plan; 4/20: integration check one; 4/24: integration check two; 4/27: deadline for final debugging.

Indexer: the indexer will have its functionality split from the bit of integration with the crawler done in HW2. This functionality remains largely unwritten. To be fully scalable, the indexer will need to implement special data structures similar to those described in section 4.2 of the Google paper. The indexer will parse downloaded pages for new URLs to tell the crawler about and calculate hits when building our search index, using the HW3 framework. The indexer also creates input for the PageRank job.

Milestones: 4/13: indexer is able to parse any new URLs from a document and add them to queue, creating PageRank input while doing so; 4/17: indexer, aided by crawler data, is able to build its index of hits; 4/20: integration check one; 4/24: integration check two; 4/27: deadline for final debugging.

PageRank: the MapReduce framework from HW3 makes implementing PageRank an easy task. Fed from shared storage of output from the indexer, this job constantly runs and updates a database storing the rankings of various URLs.

Milestones: 4/13: implementation of PageRank using test input; 4/17: data from crawler/indexer available to debug; 4/20: integration check one; 4/24: integration check two; 4/27: deadline for final debugging.

User Interface: the user interface should be simple to use and not overtly hideous. It will be based in a servlet and handle front-end tasks such as assigning worker threads to handle requests to proper locations in storage and displaying search results.

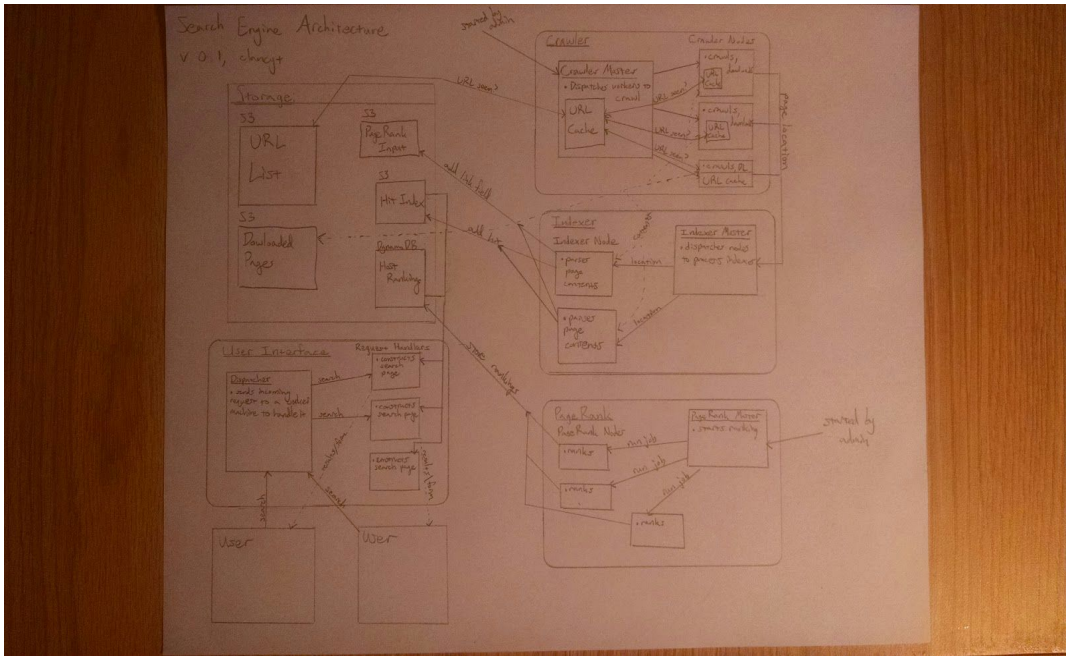
Milestones: 4/13: help the indexer; 4/17: use HW1 techniques to display indexed and ranked results; 4/20: integration check one; 4/24: integration check two; 4/27: deadline for final debugging.

Integration

Check One: The major task of this integration check is to link the indexer and crawler. Up to this point, the crawler is responsible for simply downloading URLs that may have changed. For testing before this point, the crawler retains some of its simple indexing functions from HW2. This milestone should see the crawler be able to read and write to shared storage with the indexer, which tells the crawler what URLs to check and handles processing.

Check Two: At this point the ranking function can be tuned to ensure quality search results, which will require heavy integration between PageRank and user interface. This check is intentionally placed before our final deadline (which is also before our expected actual deadline) to allot time for solving unexpected issues. Once this check is complete, group members who have completed debugging will begin the final analysis task.

Final Analysis: The requirements for this task are well-documented in the project handout.



Is this a good design?

The answer to this question will be largely determined by the results seen during our final analysis. However, we believe this design will yield good results. Following the best techniques detailed in the two technical descriptions of search engines, as well as scalability ideas from this class, the entire system is built to be distributed. The framework developed in HW3 is highly scalable and used in most components of this project. The natural bottleneck of this framework--the lone master servlets--are still an issue in this design. Given the time, we plan to implement redundancies to ensure that even master failure does not stop operation. There are two primary steps that must be very fast: retrieval of hits from user query, so that the user can see results immediately, and checking storage to determine whether a URL has been seen or not, to keep the crawler running quickly. The first will be accomplished using efficient data structures and storage, while the second will be solved using a caching on nodes, in the master server, and finally on disk during a crawl.