Search Engine Analysis
Team #2 - Semester

***Abstract*—This document shows our analysis of the performance of our Search Engine and how we performed them.**

I. The Experiment

A. Running One Experiment

This command runs the whole server in analysis mode and closes it after finishing the experiment.

```
$ env PAM=1 mvn
```

B. PerfAnalyser

`PerfAnalyser.java` is the java class responsible for conducting one experiment and closing the server afterward. It does the following:

1) Launches `$TOTAL_THREADS` of threads, each calls `Query Processor` with a random query.
2) After timeout of `$TIMEOUT_MS`, `PerfAnalyser.java` interrupts threads that didn't finish, then collects the time of the rest of the threads.
3) Calculates the average time of all threads, and calculates the number of timeouted threads.
4) Repeats this experiment one time again with the ranker disabled.
5) Queries the size of all crawled documents and the number of indexed keywords.
6) Serializes all the collected data into json file whose name follows the pattern `{performance-analysis-${TIME}.json}` and saves it into current working directory.

C. Repeating

You need to conduct this experiment multiple times during different stages of search engine running. Then plot the results to be able to answer the performance questions.

To plot the results with the python script:

```
$ python3 plot.py $PWD perf*.json
```

II. Results

Setting `$TOTAL_THREADS` = 200, `$TIMEOUT_MS` = 2 Minutes and running `PerfAnalyser.java` 10 times at different stages of database building. We got 2 figures, Fig. 1, and a flat line at 1 for the 'Success Precentage vs. Crawled Pages and Indexed keywords.'

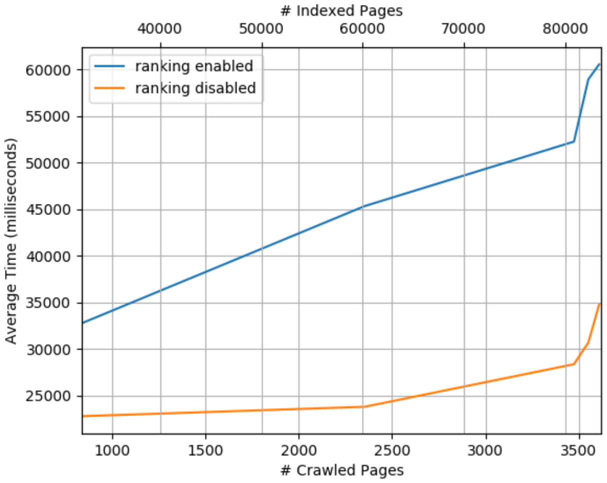

Fig. 1: Average Time vs. Num. Crawled Pages and Indexed keywords

III. Discussion

After experimenting with query processor through a range of crawled pages up to 4000 and indexed words up to 80000+, we notice that the speed of query processor both with and without ranking is affected by the database size. However, the success rate is always one, as the query processor never fails with up to 200 simultaneous requests. This is due to the following:

- In our implementation, There are multiple threads that run in parallel for crawler, indexer, popularity ranker and query processor. This results in a huge amount of database queries from all parallel threads. Consequently, by increasing the size of database, the size of query results increases, which, in return, increases the number of database access in both query processor and ranker slowing down the query response.
- We use *CoreNLP* library in the query processor, which consumes a lot of time performing `NER` *(Named-Entity Recognition)* on each query string.

IV. Conclusion

In conclusion, we can state that the main downsides about the performance of our implementation are *excess number of database access from all components at the same time* and *the usage of external libraries for some specialized tasks.*