York University at TREC 2005: Terabyte Track

Mladen Kovacevic[1] and Xiangji Huang[2]

[1]Department of Computer Science & Engineering
York University, Toronto, ON, Canada

[2]School of Information Technology
York University, Toronto, ON, Canada

Abstract

York University participated in the terabyte track this year. Using the GOV2 collection, we used filtering techniques to shorten the amount of data to be indexed before indexing into eight partitions. As there were several different subsections of the terabyte track, we chose to participate in the ad hoc and named page retrieval runs. Our technique involved partitioned indexes across a single machine. We combined our results by first calculating the document frequency of a term across all the indexes, calculating the weight, then using the same weight in retrieving the top results from each index. This approach effectively tried to mimic the results that would be obtained if there were only one large index.

1 Introduction

Sparked by the proposed methods of building significantly larger test collections by using pooling (Karen Sparck Jones and Keith van Rijsbergen), this was the second year running for the terabyte track at TREC. Continuing from last year, new evaluation methodologies are being investigated to find appropriate methods for large scale document collections. The main research goal is finding where current measures break down and exploring new ones [2]. This year, in addition to adhoc retrieval of the large collection, efficiency was explored along with named page retrieval. We had the ability to participate in the ad hoc and the named page retrieval tasks this year using the okapi-pack 2.41 information retrieval system. Preprocessing the ~480 GB GOV2 data set (the same data set given in TREC 2004), shortened the entire collection to only ~100 GB of data, without losing any significant data.

We decided to setup a partitioned environment as it seemed appropriate for this track, especially when considering that most of the time, this type of data would need to be distributed in the real world. Disk space could be a major factor for the industry. Working in parallel also allows for synchronous processing, resulting in better efficiency. In our particular setup, we were looking for ways of making the distributed environment work as if it were not distributed at all. In other words, as if there was only one large index containing all the data. Given this, the set was divided into eight partitions, thus eight indexes were created. After calculating the global document frequency of terms in each given query, the global weights were calculated using the BM25 probabilistic weighting model, which were then passed to the system to retrieve the top 10,000 documents from each index (if possible). This allowed us to make the system work in a distributed index environment, and effectively combining results as if there was only one large index.

2 Experimental Setup

2.1 System Setup

The okapi-pack system we used was version 2.41 as it contained some additional functionality to work with large data sets better. Written in C, we felt that developing an easier to use Java interface would help, thus we built an interface implemented with JNI. The new JNI interface we developed allowed us to control the okapi-pack IR system more effectively when dealing with the partitioned indexes. It made all the runs automatic and allowed us to combine the results in a simple manner.

2.2 Indexing

The indexing system in okapi-pack requires data to be first converted into a specially formatted file called *exchange format*. Prior to doing this, we used an existing information retrieval system, Terrier, from the

University of Glasgow, altered it, so as to only use its HTML parser to grab all the actual terms in the documents, and writing them to disk in exchange format. This approach removed all concepts of paragraphs in the collection, and each document simply became a *bag* of words. The indexes were then to be created based on these “bags” of words in the documents.
Machine limitations would not allow us to create a single large index. With 1.7 TB of disk space, Pentium 4 2.8 GHz processor and 1 GB of RAM, we were still limited and thus were forced to create smaller indexes. At first, the exchange files, before being indexed, were divided equally by number of documents. The entire collection contained ~25 million documents, so we divided the file to contain ~5 million documents in each exchange file. Indexing still failed on the first partition and the fifth partition, because, even though there were the same number of documents, the sizes of the documents were larger. We further divided the first exchange file by two, and divided the fifth exchange file by three. Index creation was then successful and we labeled the indexes accordingly.

Each of the indexes were created by first removing stop-words, then stemming each of the words using Porter’s stemming algorithm. This allowed for the smallest indexes we could make, while maintaining all the relevant information in tact. We wanted to make sure we’re using the full data set, and we note that further improvements could have been made using compression mechanisms, however, this would also add hindrance to query performance.

2.3 Querying

Querying was performed in a two step process. Given the topic file of specific forms (different for all three subsections of the terabyte track), we first extracted all the terms from the various fields available, removed stop-words, and stemmed each of the terms using the same Porter’s stemming algorithm used in indexing. We then kept a count of occurrences of each of the terms in the query, then sorted them. This allowed us to run tests by running queries with only the top *n* query terms used in the search. After this, the terms were passed to the okapi-pack system, and document frequency was fetched from each of the indexes. The sum of the document frequencies retrieved was taken, and consequently used in the BM25 weighting model, calculating each term’s weight based on the *entire* collection. Given the global weights for each of the terms, the top 10,000 documents were retrieved from each index separately. The global 10,000 results written to the final output file, were simply taken one by one from highest to lowest scores recorded by each of the indexes. In theory, if one index had all its results containing scores higher than all the scores found in other indexes, the final results would then contain all the results from this one index. That is why it was necessary for each index to return the required 10,000 results for each query given by TREC.

2.4 Environments for Submitted Runs

There were two runs that we submitted for the terabyte track. They were the ad hoc retrieval and the named page retrieval tasks, named tera05tAa1.run and tera05tNa1.run respectively.

2.4.1 Ad hoc Terabyte Run

The ad hoc run for the terabyte track was similar to the run last year. The run consisted of 50 queries, run against the system either automatically (with no human intervention other than setup), or manually (with some human intervention, i.e. choosing appropriate words from query). We submitted one run, under the name, york05tAa1.run. Each of the queries given consisted of three fields, “title”, “description”, and “narrative”. In our run, we grabbed terms from all three fields, and processed them according to the description given above. At the time of the ad hoc runs, the final index, tera05_5c, was not yet successfully created, thus we only used just over 90% of the collection for this run. The machine we used for this run was a Pentium 4, 2.8 GHz, with 1.7 TB of disk space and 1 GB of RAM.

2.4.2 Named Page Run

The named page run for the terabyte track was new this year, with a primary focus on retrieving the page with the corresponding given title from the query. The results were *not* measured by score of the document, but rather the *rank* in which the information retrieval system found the required document.
The query file contained a total of 272 topics, structured by the number of the query, followed by the title to be searched. Queries were processed the same as described above, and the same search was performed across all the partitioned indexes. The difference in this case was that we were able to get all 8

indexes working properly, thus, indexing 100 % of the dataset. Our run was labeled as tera05tNa1.run. The machine we used in this scenario was different than the Ad hoc run, where we had a Pentium 4, 2.8 GHz dual processor, with 250 GB disk space, and 2 GB of RAM.

3 Experimental Results and Analysis

3.1 Indexing

Indexing was needed to be performed in a few distinct steps; meaning that the collection could not be simply given to the okapi-pack system and it would be indexed. Currently, okapi-pack only supports reading a certain type of format of files, called exchange files, thus the first step was to convert the collection into this type of file. This process took an overwhelming amount of time, which was not measured accurately because of the amount of hardware problems we were having with the system. Another problem here was the fact that this processing was done in Java, and it seems that memory was a large concern, causing the process to fail on various occasions.
After the data was prepared in exchange format, okapi took over from there and from then on it took between 5 – 7 days to complete the indexing process.

3.2 Querying

The method used for querying was an attempt to simulate a pure distributed information retrieval system. Our limited machine capabilities stopped us from developing an actual distributed environment where the queries could be run at the same time on each partition then simply merge the results. In this case, we had the indexes partitioned on the same system, thus, in actuality we performed the queries in series, passing through all the indexes twice. Each time we access an index, we actually need to “connect” to the index first, ask the okapi-pack system to give us results, then “connect” to the next. With eight partitions, connecting to each of the indexes proved to be an enormous bottleneck when evaluating query performance (in terms of speed of retrieval).

Our experiments were focused on making the partitioned indexes work as though they were actually a single large index. In doing this type of work, clearly, we sacrificed query retrieval time as we were not able to test in our current environment.

3.3 Submitted Runs

3.3.1 Ad hoc Run

Our ad hoc terabyte run for the 50 topics provided by TREC 2005, earned a mean average precision value (MAP) of 0.1565. Thirteen of the fifty queries returned extremely low results, below 0.02 mean average precision values. In the top five results, we achieved mean average precision values of over 0.40. The extremely low results could very well be from the fact that we were unable at the time to index the *entire* dataset (only ~90%). Evaluation results were provided by TREC directly who used the “trec_eval” program version 7.3. The “trec_eval” program was set to evaluate the top 1000 documents, although our submitted results contained the top 10,000 retrieved documents. Further analysis as to why we achieved such low results still needs to be performed.

As far as retrieval performance is concerned, we performed evaluation by only retrieving the top 20 results for each query (instead of the top 10,000 as required by TREC). All query fields were analyzed, parsed and passed to okapi-pack automatically (on all three fields; title, description, narrative). Each query took an average of 161 seconds, with a total time of 8056 seconds for the fifty queries.

Here is a summary of our results:

Submitted run results, querying all fields of each topic, and complete automatic retrieval

Run	**MAP**	**R-prec**	**bpref**	**Recip_rank**	**P5**	**P10**	**P15**
Trec05tAa1.run	0.1565	0.2148	0.1816	0.5951	0.4240	0.4120	0.4040

3.3.2 Named Page Run

The named page run had a different purpose in that the *rank* at which we found the particular correct page was looked at. Therefore, the scores given for each document were not relevant, rather only the *rank* at which the correct document was placed in. There were a total of 272 queries, however, for 30 of the topics, *no* run was able to find correct answers. Twenty of the topics were removed as they were not considered to be proper named-page topics.

Named Page run results

Run	Reciprocal Rank (252 topics)	# topics page found in top 10	# topics no page found
York05tNa1.run	0.329	112 (44.4%)	64 (25.4%)

The methods used in the named page run were no different from the approach taken for the ad hoc run. Clearly, this approach could be improved. Several methods that we could take are by indexing the entire collection based only on the <title> html flag in the documents. However, as shown in [1], it has been found that about 1/3 of pages in the .GOV domain have no meaningful titles. For query retrieval time, however, this approach could still be tested, but perhaps expanded. In the event that no pages are found given the title index, we could then, and only then, resort to the full index containing all words in the document. This approach has not yet been attempted with our systems.

In our tests, the average query time per topic was 42 seconds. The total amount of time taken was 11,495 seconds for all 272 topics. Once again, we feel the largest bottleneck is simply because of the multiple connections we need to make to different indexes in a serial environment. We believe that these retrieval times should reduce dramatically if this system was implemented in a true distributed environment.

4 Conclusions

The most significant approach we took this year in participating in the terabyte track at TREC 2005 included building a distributed index system, using mechanisms to retrieve the highest scored documents as if we were running using a single large index. We did this by performing a two pass run through all eight partitions for each query. The first pass calculated the term document frequency for all terms given in the query, while the second run took those results, calculated the weight of the terms using the BM25 weighting model, and applied these results in the retrieval for each index. Thus, the calculated weight of each term was the same weight used across all partitions, mimicking a single large index.
Performance results in terms of retrieval times proved to be quite slow, however, this was assumed given the fact that we did not have the capability to use multiple machines in the distributed index environment. Almost half of the topics in the named page run were found within the top ranked results, however, we feel that this could be improved by developing new techniques specific for named page results (building an index specific to this area). The ad hoc task produced results with a MAP of 0.1565, which we also feel we could improve upon. The submitted results also did not include our run against 100% indexed database which may have played an important role in lowering our results. More methods still need to be investigated and tested using the okapi-pack system.

5 Acknowledgements

We would like to thank the Terrier development team for their support in using their software, along with Andrew MacFarlane for giving input in resolving issues with okapi-pack.

6 References

[1] R. Song, J.R. Wen, S. Shi, G.Xin, et al. (2004). “Microsoft Research Asia at Web Track and Terabyte Track of TREC 2004”, In D.K.Harman, editor, Proceedings of the Thirteenth Text Retrieval Conference (TREC-13), Gaithersburg, MD, NIST Special Publication.

[2] C. Clarke, N. Craswell and I. Soboroff (2004) “Overview of the TREC 2004 Terabyte Track”, In D.K.Harman, editor, Proceedings of the Thirteenth Text Retrieval Conference (TREC-13), Gaithersburg, MD, NIST Special Publication.