# MapReduce for Experimental Search

Djoerd Hiemstra and Claudia Hauff

University of Twente, The Netherlands
{d.hiemstra, c.hauff}@utwente.nl

**Abstract**

This draft report presents preliminary results for the TREC 2010 ad-hoc web search task. We ran our MIREX system on 0.5 billion web documents from the ClueWeb09 crawl. On average, the system retrieves at least 3 relevant documents on the first result page containing 10 results, using a simple index consisting of anchor texts, page titles, and spam removal.

## 1 Introduction

Information retrieval researchers are supposed to come up with new ideas that improve search systems. If such ideas are radically new, experimentally testing might require a non-trivial amount of coding to change existing systems. If for instance a new idea requires information that is not normally in the index (anchor texts, geographical locations, some complex natural language constructs, parsimonious language modeling probabilities, etc., etc.), then the researcher has to recode parts of the system's indexing facilities, and possibly query processing facilities that access this information. If the new idea requires query processing techniques that are not supported by the production system (sliding windows, phrases, structured query expansion, ontology matching, etc.) even more work has to be done.

We propose to use Hadoop MapReduce [6] to quickly test new retrieval approaches on a cluster of machines by sequentially scanning all documents. The system read each web page one at the time, and on each page we execute the 50 TREC topics. Sequential scanning allows us to do almost anything we like: sophisticated natural language processing, sliding windows, etc. If the new approach is successful, it will have to be implemented in a full blown inverted file search system, but there is no point in making a new inverted index if the experiment is unsuccessful. To give the reader an idea of the complexity of such an experiment: An experiment that needs two sequential scans of the data requires about 350 lines of code. The experimental code does not need to be maintained: In fact, it should not be altered anymore to provide data provenance and reproducibility of research results. Once the experiment is done, the code is filed in a repository for future reference. We call our code repository MIREX

(MapReduce Information Retrieval EXperiments), and it is available as open source software from `http://mirex.sourceforge.net`

Surprisingly, linear scanning for testing experimental search approaches seems to be mostly reported in industrial settings. Jeffrey Dean [3] describes how MapReduce [4] is used at Google for experimental evaluations. New ranking ideas are tested off-line on human rated query sets similar to topics at TREC. Running such off-line tests has to be easy for the researchers at Google, possibly at the expense of the efficiency of the prototype. So, it is okay if it takes hours to run for instance 10,000 queries, as long as the experimental infrastructure allows for fast and easy coding of new approaches. A similar line of reasoning was followed by Microsoft at TREC 2009: Nick Craswell et al. [2] use DryadLINQ [7] on a cluster of 240 machines to run their TREC experiments. The work at Google and Microsoft shows that sequential scanning over large document collections is a viable approach to experimental information retrieval. Some of the advantages are [5]: 1) Researchers spend less time on coding and debugging new experimental retrieval approaches; 2) It is easy to include new information in the ranking algorithm, even if that information would not normally be included in the search engine's inverted index; 3) Researchers are able to oversee all or most of the code used in the experiment; 4) Large-scale experiments can be done in reasonable time.

This draft report presents preliminary results for the TREC 2010 ad-hoc task. In Section 2 we briefly introduce MIREX, Section 3 presents our experimental results, and Section 4 concludes the paper.

## 2 MIREX: MapReduce Information Retrieval Experiments

MapReduce is a framework for batch processing of large data sets on clusters of commodity machines [4]. Users of the framework specify a *mapper* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reducer* function that processes intermediate values associated with the same intermediate key. The pseudo code in Figure 1 outlines our sequential search implementation. The implementation does a single scan of the documents, processing all queries in parallel.

The *mapper* function takes pairs of *document identifier* and *document text* `(DocId, DocText)`. For each pair, it runs all benchmark queries and outputs for each matching query the *query identifier* as key, and the pair *document identifier* and *score* as value. In the code, `Queries` is a global constant per experiment. The MapReduce framework runs the mappers in parallel on each machine in the cluster. When the map step finishes, the framework groups the intermediate output per key, i.e., per `QueryId`. The *reducer* function then simply takes the top 1000 results for each query identifier, and outputs those as the final result. The reducer function is also applied locally on each machine (that is, the reducer is also used as a *combiner* [4]), making sure that at most

```
mapper (DocId, DocText) =
  FOREACH (QueryID, QueryText) IN Queries
    Score = experimental_score(QueryText, DocText)
    IF (Score > 0)
    THEN OUTPUT(QueryId, (DocId, Score))

reducer (QueryId, DocIdScorePairs) =
  RankedList = ARRAY[1000]
  FOREACH (DocId, Score) IN DocIdScorePairs
    IF (NOT filled(RankedList) OR
      Score > smallest_score(RankedList))
    THEN ranked_insert(RankedList, (DocId, Score))
  FOREACH (DocId, Score) IN RankedList
    OUTPUT(QueryId, DocId, Score)
```

Figure 1: Pseudo code for linear search

1000 results have to be sent between machines after the map phase finishes.

Our experiments are made of several such MapReduce jobs: We extracted anchor texts from the full ClueWeb09 Category A data, we gathered global statistics for terms that occur in the TREC topics, and we run linear search as depicted in Figure 1. As a final step, we removed spam pages using the results from Gordon Cormack et al. [1]. The experiments were run on a cluster of 15 low cost machines.

## 3 Experimental results

Table 1 shows the precision at 5, 10 and 20 web pages retrieved for queries 1–50 of TREC 2009. Here, LM no smoothing denotes a language model with a length prior, that does not use any smoothing such that documents that do not contain all query terms are assigned zero probability (i.e., they are not retrieved). The experiment named LM $\lambda = x$ denotes a language model with length prior and linear interpolation smoothing, where $\lambda$ is the weight of the document language model, so $\lambda = 0.95$ uses minimal smoothing, and $\lambda = 0.15$ uses quite some smoothing. The experiment named LM Dirichlet uses Dirichlet smoothing with $\mu = 2500$, and BM25 is Okapi's BM25 weighting with $k_1 = 1.2$ and $b = 0.75$ [5].

Interestingly, on the anchor text representation, the best approach does not use smoothing, so it does not need global statistics to compute IDF-like (inverse document frequency) weights. Global statistics can be incorporated by doing one initial pass over the corpus to collect global statistics for all queries [2]. Presumably, on document collections of this scale, IDF-like weighting is unnecessary.

| Experiment | P@5 | P@10 | P@20 |
|---|---|---|---|
| anchors, LM no smoothing | 0.372 | 0.330 | 0.253 |
| anchors, LM $\lambda = 0.95$ | 0.372 | 0.328 | 0.257 |
| anchors, LM $\lambda = 0.15$ | 0.312 | 0.276 | 0.233 |
| anchors, LM Dirichlet | 0.316 | 0.282 | 0.214 |
| anchors, BM25 | 0.312 | 0.246 | 0.200 |
| anchors+titles, $\lambda = 0.95$ | 0.360 | 0.316 | 0.251 |
| anchors–spam, LM $\lambda = 0.95$ | 0.416 | 0.338 | 0.261 |
| anchors+titles–spam, LM $\lambda = 0.95$ | 0.412 | 0.332 | 0.257 |

Table 1: Precision at 5, 10 and 20 on TREC 2009

We used the spam rankings kindly provided by Cormack et al. [1] – their *fusion* results – to remove the spammiest 50% of the web pages, i.e., we effectively removed half of the index. Removing spam pages helps early precision considerably on TREC 2009, although not as much as reported by Cormack et al. Searching the document titles in addition to the anchor texts seems to hurt performance a bit on TREC 2009, however, it improved recall (not shown), and in combination with spam page removal, the precision figures are almost the same as searching the anchor texts alone.

| Experiment | P@5 | P@10 | P@20 |
|---|---|---|---|
| anchors, LM $\lambda = 0.95$ (*utwente3*) | 0.306 | 0.256 | 0.251 |
| anchors+titles, LM $\lambda = 0.95$ (*utwente4*) | 0.311 | 0.253 | 0.249 |
| anchors–spam, LM $\lambda = 0.95$ (*unofficial*) | 0.317 | 0.306 | 0.276 |
| anchors+titles–spam, LM $\lambda = 0.95$ (*utwente4SF*) | 0.317 | 0.308 | 0.303 |

Table 2: Precision at 5, 10 and 20 on TREC 2010

Table 2 shows the precision at 5, 10, and 20 documents retrieved of the official TREC runs. The results are computed over 36 of the 50 topics for which relevance judgments were available at the time of writing these working notes. The held back topic numbers are 54, 61, 66, 68, 72, 78, 83, 86, 87, 90, 95, 98, 99, and 100. The official runs are fully judged until 20 documents retrieved, so all precision values are actual precision values. Of the unofficial run, the fraction of judged documents for 5, 10 and 20 documents retrieved was 1.00, 0.99, and 0.95 respectively. On TREC 2010, including the document titles seems to help a bit, even if spam pages are not removed. The three official runs do not differ substantially on the 36 topics.

## 4 Conclusion

We showed how to use Hadoop MapReduce for large-scale information retrieval experiments using little effort. We evaluated a web search approach that uses anchor texts, page titles, and spam detection. If we assume the standard result page size of 10 results, then the system retrieves more than 3 relevant documents on average on the first result page. The code used in our experiment is open source and available to other researchers at: `http://mirex.sourceforge.net`

## Acknowledgements

The research was funded in part by the Netherlands Organization for Scientific Research (NWO grant 639.022.809). We are grateful to Yahoo Research, Barcelona, for contributing to our Hadoop cluster.

## References

[1] G. Cormack, M. Smucker, and C. Clarke. Efficient and effective spam filtering and re-ranking for large web datasets. *arXiv:1004.5168*, 2010.

[2] N. Craswell, D. Fetterly, M. Najork, S. Robertson, and E. Yilmaz. Microsoft Research at TREC 2009: Web and relevance feedback tracks. In *Proceedings of the 18th Text REtrieval Conference (TREC)*. NIST Special Publication 500-278, 2009.

[3] J. Dean. Challenges in building large-scale information retrieval systems: invited talk. In *Proceedings of the second ACM International Conference on Web Search and Data Mining (WSDM)*, page 1, 2009.

[4] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implemention (OSDI)*, 2004.

[5] D. Hiemstra and C. Hauff. MapReduce for information retrieval evaluation: let's quickly test this on 12 TB of data. In *Multilingual and Multimodal Information Access Evaluation*, Lecture Notes in Computer Science 6360, pages 64–69. Springer Verlag, 2010.

[6] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, 2009.

[7] Y. Yu, M. Isard, D. Fetterly, M. Budiu, U. Erlingsson, P. Kumar, and J. Currey. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In *Proceedings of the 8th Symposium on Operating System Design and Implemention (OSDI)*, 2008.