On Single-Pass Indexing with MapReduce

Richard M. C. McCreadie
Department of Computing Science
University of Glasgow
Glasgow, G12 8QQ
richardm@dcs.gla.ac.uk

Craig Macdonald
Department of Computing Science
University of Glasgow
Glasgow, G12 8QQ
craigm@dcs.gla.ac.uk

Iadh Ounis
Department of Computing Science
University of Glasgow
Glasgow, G12 8QQ
ounis@dcs.gla.ac.uk

ABSTRACT

Indexing is an important Information Retrieval (IR) operation, which must be parallelised to support large-scale document corpora. We propose a novel adaptation of the state-of-the-art single-pass indexing algorithm in terms of the MapReduce programming model. We then experiment with this adaptation, in the context of the Hadoop MapReduce implementation. In particular, we explore the scale of improvements that can be achieved when using firstly more processing hardware and secondly larger corpora. Our results show that indexing speed increases in a close to linear fashion when scaling corpus size or number of processing machines. This suggests that the proposed indexing implementation is viable to support upcoming large-scale corpora.

Categories and Subject Descriptors: H.3.3 [Information Storage & Retrieval]: Information Search & Retrieval

General Terms: Performance, Experimentation

Keywords: Indexing, MapReduce

1. INTRODUCTION

With the ever-growing test corpora being developed for researchers, scalable implementations of common IR operations have become a necessity. Indeed, TREC in 2009 increased the size of its largest collection by almost 12 times. As MapReduce has gained popularity in commercial settings, with implementations by Google [3], Yahoo! [2] and Microsoft [5], it seems likely to be a viable tool for large-scale data processing. To investigate this, we propose development of a state-of-the-art indexing operation, in the context of MapReduce. While the original MapReduce paper [3] shows the scalability of MapReduce in general, it does not give an in-depth explanation of how efficient indexing should be performed within such a model.

In Section 2, we develop a novel indexing strategy in MapReduce, inspired by the single-pass indexing of Heinz & Zobel [4]. In particular, we employ Hadoop [2] - the only freely available MapReduce implementation, developed by Yahoo!. Section 3, presents experiments using our MapReduce indexing strategy, to assess its scalability, in terms of hardware and input data. The main contributions of this work are three-fold: we show how the MapReduce paradigm can be successfully applied to an existing IR system and provide a working implementation to the community; we provide a working example of MapReduce of significantly greater complexity than the commonly-used MapReduce word count example; and we experiment to show the benefits and scalability of the proposed implementation.

SIGIR'09, July 19–23, 2009, Boston, Massachusetts, USA.
ACM 978-1-60558-483-6/09/07.

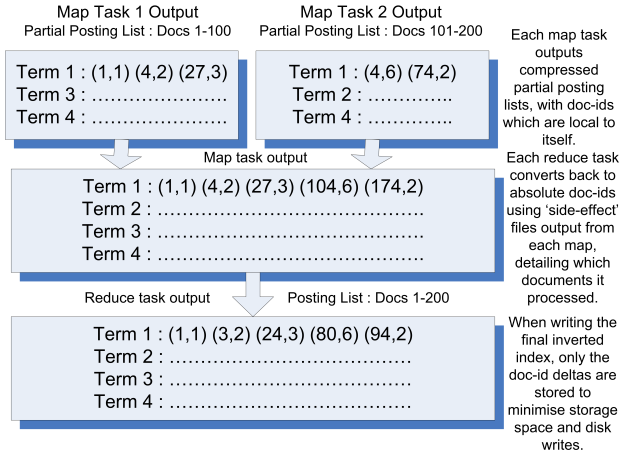

Figure 1: Correcting document IDs while merging.

2. HOW-TO IMPLEMENT SCALABLE INDEXING IN MAPREDUCE

We adapt the state-of-the-art indexing strategy - single-pass indexing [4] for MapReduce. In single-pass indexing, (compressed) posting lists for each term are built in memory as the corpus is scanned. When local memory is exhausted, the partial indices are 'flushed' to disk and the final index is built from the merged flushes. Elias-Gamma compression is used to store document-identifier (doc-id) deltas, ensuring postings are fully compressed, both in memory and on disk.

We now propose a conversion for single-pass indexing into MapReduce. Document processing is split over m map tasks, with each map task processing its own subset of the input data. However, when memory runs low or all documents for that map have been processed, the partial index is flushed from the map task, by emitting a set of <term, posting list> pairs. As in single-pass indexing, the posting lists are compressed to minimise the data that is transferred between map and reduce tasks. Moreover, as each map task is not aware of its context in the overall indexing job, the doc-ids used in the emitted posting lists cannot be globally correct. Instead, these doc-ids start from 0 in each flush.

The partial indices (flushes) are then sorted by term, map and flush numbers before being passed to one or more reduce tasks. Each reducer collates the posting lists to create the final inverted index for the documents it processed. In particular, as the flushes are collected at an appropriate re-

duce task, the posting lists for each term are merged by map number and flush number, to ensure that the posting lists for each term are in a globally correct doc-id ordering. The reduce function takes each term in turn and merges the posting lists for that term into a full posting list. Figure 1 presents an example for a distributed setting MapReduce indexing paradigm of 200 documents. Note that the number of reduce tasks therefore determines the final number of inverted index shards created.

3. EXPERIMENTATION & RESULTS

To determine the extent to which MapReduce is a suitable framework for efficiently processing large IR corpora, we investigate two research questions: does our Hadoop MapReduce implementation attain linear speedup with machines allocated (i.e. doubling machines would ideally half indexing time); and how does corpora size affect performance? Our indexer uses the Hadoop MapReduce implementation (v. 0.18.2) and we evaluate using four standard TREC corpora of varying size, namely WT2G, WT10G, .GOV and .GOV2. Of these, .GOV2 is the largest at 25M documents, comprising 425GB when uncompressed.

Firstly, we test to determine if the distributed (MapReduce) indexing will complete the same indexing process in a shorter time as we increase the processing power available. To show this, we measure the mean indexing time of .GOV2 (5 repetitions), running on 1-8 machines, when using a single reduce task. From the single reducer curve in Figure 2, we observe that indexing time decreases as more machines are added (i.e. speedup increases). However, by examining the trends observed as the number of machines increases, we see that *linear* speedup is not achieved, as indexing time speedups level off after approximately 6 machines.

On further analysis, we suggest that this is due to the use of a single reduce task, with this becoming the bottleneck of the indexing job, i.e. the (sequential) single reduce task limits the speedup achievable as described in Amdahl's law [1]. To test this, we then experimented indexing when using 24 reducers. The results are presented in the multiple reducer curve in Figure 2. Indeed, this shows that by using multiple reduce tasks we achieve marked performance improvements beyond 6 machines. As an illustration to this success, we note that the single-pass (single-threaded) indexing took over a day (1605 minutes) to index .GOV2. However, when running the multi-threaded MapReduce implementation on a single three-core machine, indexing completed in less than 8 hours (472 minutes), while for 8 machines this is reduced to just over an hour (73 minutes). This represents a 6.5 times speedup for the MapReduce implementation between 1 and 8 machines.

However, as this is still sub-linear scaling, we further suggest that this can be explained in terms of a lack of file locality to the machines doing the work. As of v. 0.18.2, Hadoop ignored file locality when assigning multiple files to each map task[1]. To investigate the impact of this, we increased the availability of .GOV2 until all machines had their own copy - thereby eliminating the need to transfer files. The results are also presented in Figure 2, which clearly shows scaling close to linear in nature. We can therefore conclude that our MapReduce indexing implementation scales with processing power in a fashion which is appropriate for efficient computation. Moreover, linear speedup can be achieved through maximisation of file locality.

[1]Later Hadoop versions have made improvements in this area.

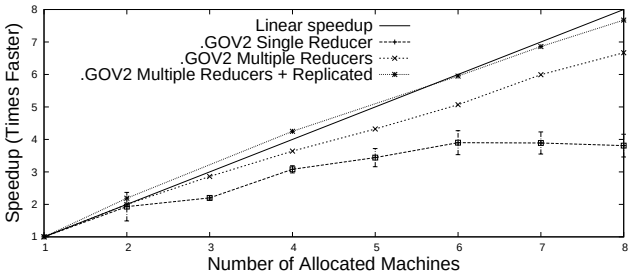

Figure 2: .GOV2 indexing speed increase curves as more machines are allocated. Single reducer experiments are repeated 5 times - error bars are shown.

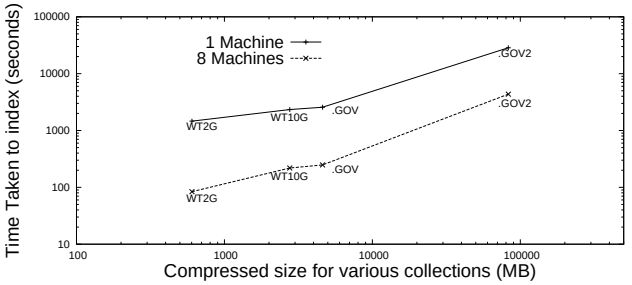

Figure 3: Indexing time for all 4 collections using both 1 and 8 machines, and 24 reduce tasks.

Next, we show that the MapReduce indexer scales well as the size of the input data increases. To test this, we index each of our corpora, using 1 machine, and 8 machines. Figure 3 presents the MapReduce indexing times (not speedup) for various compressed sizes of corpus. From this figure, we observe that indexing of all corpora takes less time using more machines. As expected, when the corpus size is increased, indexing takes longer, however, the general trends of the curves are slightly convex in nature, indicating that scaling with corpus size is marginally sub-linear.

4. CONCLUSIONS

In this paper we have shown how to distribute a common IR task within the MapReduce paradigm, namely indexing. Our results show that indexing could be successfully distributed across a cluster of machines, using the Hadoop MapReduce framework. Moreover, we show that the MapReduce indexing implementation is well suited for processing of large-scale collections as its performance scales close to linearly with processing power and collection size. The implementation described in this paper is freely available for use by the community as part of the Terrier IR Platform[2].

5. REFERENCES

[1] G. Amdahl. Validity of the single processor approach to achieving large-scale computing capabilities. In *Proceedings of AFIPS*, pgs. 483–485, 1967.

[2] Apache Software Foundation. The Apache Hadoop project. `http://hadoop.apache.org/`, accessed on 25/01/2009.

[3] J. Dean and S. Ghemawat. Simplified data processing on large clusters. In *Proceedings of OSDI 2004*, pgs. 137–150.

[4] S. Heinz and J. Zobel. Efficient single-pass index construction for text databases. *JASIST*, 54(8):713–729, 2003.

[5] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of EuroSys 2007*, pgs. 59–72.

[2]`http://terrier.org`