Parallel Methods for the Search of Partitioned Inverted Files

[*]A. MacFarlane, [*+]S.E.Robertson, [*]J.A.McCann
[*]School of Informatics, City University, London EC1V OHB
[+]Microsoft Research Ltd, Cambridge CB2 3NH

Abstract: We examine the search of partitioned Inverted Files with particular emphasis on issues which arise from different types of partitioning methods. Two types of Index Partitions are investigated: namely *TermId* and *DocId*. We describe the search operators implemented in order to support partitioned search. We also describe higher level features such as search topologies and relevance feedback in partitioned search. The results from runs on both type of partitioning are compared and contrasted.

1. INTRODUCTION

In this paper we describe search mechanism for PLIERS, a parallel Information Retrieval system based heavily on ideas developed on the Okapi system at City University [1]. Our aim in this research is to examine the issue of Search on Inverted files given two types of partitioning methods: Term Identifier (*TermId*) Partitioning and Document Identifier (*DocId*) partitioning. Term Identifier partitioning is a type of partitioning which distributes a unique word to a fragment, while Document Identifier partitioning distributes a unique document to a fragment. A fragment is a physical division of the Inverted file. A fuller discussion of these partitioning methods can be found in [2,3]. The experimental aims and objectives of our research is given in section 2 which is overall to examine which Inverted file partitioning method provides the best retrieval efficiency results. Search operation functionality provided by PLIERS is described in section 3. The issue of search topologies is discussed in section 4, while section 5 gives a technical description of how the operations are implemented on different partitioning methods. How relevance feedback is supported on partitioned inverted files is discussed in section 6. The hardware used in the experiments is outlined in section 7. The data and settings used for the experiments are described in section 8. The results of searches on the two chosen partitioning methods is reported in sections 9 and 10, and the conclusion in section 11 compares and contrasts these results with our stated aims/objectives.

2. EXPERIMENTAL AIMS AND OBJECTIVES

The previous work on the subject of search performance on Partitioned Inverted files [2,18] used simulations in order to do the research. We attempt to use real collections using TREC queries to examine performance. The query model used in these simulations assumed either an equal probability of word occurring in a query [18] or did not address the issue [2]. We believe the distribution of words in Queries to be of fundamental importance with respect to partitioning methods and we wish to examine a Hypothesis: performance on Indexes with DocId partitioning will on average yield a better and more predicable performance than TermId partitioning because of the Zipt distribution.

This Hypothesis is quite abstract and needs further elaboration. Much depends on users information needs: users formulate their queries on the basis of this information need by choosing terms they regard as useful for retrieval [20]. User actions have a direct impact on search performance or to put it another way performance is not independent of user action. User action determines the distribution of words in the query. This is true of all IR systems, but becomes very important given a fragmented index. Since users information needs may change over time it may not be possible to produce a distribution of words in queries and there is no general agreement on word distribution in queries [18]. To estimate the distribution of terms in Queries we would need to know every potential term set for every Information need. Our assumption is that this is for the most part an unknowable variable and seek ways of understanding search performance in the absence of such data. We do this through an examination of Zipf's law [19] which informally states: a few words occur many times in a document collection while many words will occur only once. We seek our answer through what we can know: the distribution of words in the collection.

The key issue is how fairly are terms distributed on average to partitions for the inverted file. In the case of DocId partitioning we assume a random distribution of document to partitions gives an approximately equal distribution of documents to partitions. We further assume that frequency of words occurrence are similar if and only if and only if the document distribution is fair (this assumption does not cover low frequency words). Given these assumptions the Zipfian distribution will roughly be the same and query performance on each partition will be similar. However with TermId partitioning

the Zipf distribution on partitions will be different. The frequency of word occurrence will not be similar over partitions by the very definition of TermId partitioning. In the worst case one partition may have all of the Query words. Therefore we hypothesise that the load balance for search on DocId partitioned indexes will be superior to those on TermId partitioned indexes on average, therefore search will be on average faster on DocId than TermId.

It should be stated that in some circumstances information on queries can be gather in post search analysis. In a paper on Internet search Kirsch [21] stated that about 12% of the queries submitted to Infoseek was by users with an interest in procreation. It may be possible to use this data to produce a Query distribution in order to reassign some of the terms to different fragments in order to improve load balance where TermId is used. We do not address the issue of dynamic migrating Indexes.

It should be noted that we address the issue of retrieval efficiency in this study, but do mention retrieval effectiveness in order to verify that searching TermId and DocId partitioned Indexes yield same results (as they should do).

3. SEARCH OPERATION FUNCTIONALITY

The search operations provided by PLIERS are described in this section. The operations discussed here are placed into three distinct classes: Boolean (AND, OR, AND-NOT), Proximity (ADJ, SAMES, SAMEP, SAMEF) and Weighting (PLUS). The model used for search is the Search Set model, therefore these operations are defined over sets. The semantics of these operations are described more fully in Robertson/Walker [7] and MacFarlane et al [8].

3.1 Boolean Operations

With Boolean operations set merges are done on the basis of document identifiers: the union of two sets are taken for the OR operation, the intersection for the AND operation, and the difference for the AND-NOT operation. A single OR operation or OR operations are simple queries: the operation is associative and has the identity Æ, the empty set.

3.2 Proximity Operations

Proximity operations on sets are defined over the position lists which are associated with each element of a inverted list. The ADJ operator looks for words in document which are adjacent to each other: in this particular case we have implemented an ordered ADJ i.e. A ADJ B != B ADJ A. The SAMES operator looks for words in the same sentence, SAMEP looks for words in the same paragraph and SAMEF looks for words in the same fields. Each of these operators is implemented as a special case of intersection, the criteria of which is defined by the position information required to satisfy that operations semantics. An appropriate function is associated with each set header. If we take SAMEP as an example: words in the same paragraph must also be in the same field therefore both field and paragraph number are checked. The implementation of the ADJ operator is slightly more complicated as order of word position is important as are any preceding stop word between the two relevant terms.

3.3 Term Weighting Operations

The term weighting model supported in PLIERS is the Robertson/Sparck Jones Probabilistic model [4,9]. We use a sum of weights method (PLUS) which requires two phases for term weighting operation: calculation of the weights and then set merge with addition of those weights. The final result set is then sorted by weight in decreasing order of weight. In the calculation of weights method we use a number of functions which have been applied at Okapi at TREC including bm_0, bm_11, bm_15 [10] and bm_25 [11] (bm means best match). All results where produced on the bm_25 term weighting function.

- do a set of searches without doclen (bm_15) to compare performance with bm_25?

Note: should I declare the formulas for the weighting function being used? (don't see point given that its well know - maybe for the thesis only?

4. SEARCH TOPOLOGIES

In order to facilitate parallel searches on Inverted files with differing data partitioned methods, we need some generic method with which to service queries. We define two components of a parallel search topology, a top node and one or more leaf nodes. Our requirements are that we should define these components so they can be re-configured irrespective of the type of data partitioning method used: the client sees a single Inverted file and retrieval responses with respect to effectiveness measures are identical on any data partitioning method. These components are described in below, followed by a discussion on search topologies.

4.1 Top Node

The main task of the Top Node in a search topology is to act as the interface for a client to the topology. It accepts queries from the client, distributes it to all of its child nodes and awaits the results. Depending on the type of operation it may sort the results ready for presentation to the client, or order merge partially ordered results from its child nodes (see section 4.3 for more discussion on this point).

4.2 Leaf Node

The Leaf Node looks after one fragment of the Inverted files. It keeps an in-core tree of keywords (as described in section 2) which is searched when a query is received. The inverted lists are then built for each element of the query and merged together to form a final result set. This result set or sets are sent to the top node. The number of leaf nodes are defined by the number of Inverted File fragments.

4.3 Discussion on search topologies

An example of how the components are combined can be found in figure 1.

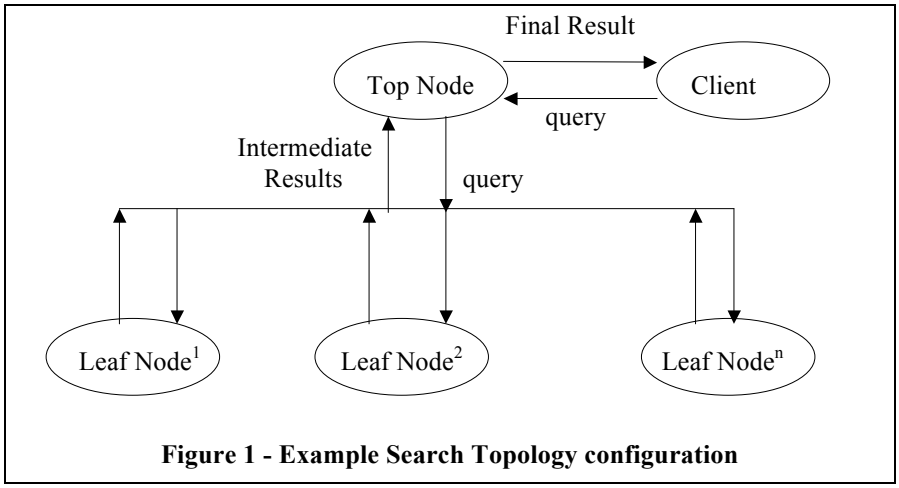

Figure 1 - Example Search Topology configuration

The example in figure 1 is a tree topology with a top node and n leaf nodes. The service of a query is done as follows: the top node receives a query and distributes it to Leaf Nodes 1 and 2 to n. The result set for that query is sent back in the inverse direction merging as necessary. [note: may need a bit more detail here- but what?]

5. SEARCH OPERATIONS ON PARTITIONED INVERTED FILES

How search operations are implemented on partitioned Inverted Files using different sorts of search topologies are the subject of this section. We define three sorts of Queries: a Simple Query which can be merged immediately, Weighted Queries which are almost identical to Simple Queries but involve the addition of weights, and Complex Queries where results may not be able to be merged until information is received by the top process. How these queries are serviced using different types of partitioning methods are of particular interest. We put the discussion on Boolean, Proximity and Weighted operation in separate sections as per section 3.

5.1 Boolean Operations

Single AND and AND-NOT operations are regarded as complex queries, AND because the identity is a non-empty set identical to it, and AND-NOT because it is not associative. Any combination of these three Boolean operations is a Complex query. The implementation for set merge on a simple query containing an OR operation is straightforward in both DocId and TermId partitioning methods: the union is applied to incoming query term sets and placed on the output link. The situation for complex queries with regard to partitioning methods is very different. With TermId partitioning the full information for a query is not available until all results have been received at the top node: some or all of the sets need to be transmitted from the leaf nodes to the top node (though any internal nodes which may be between them). However with DocId partitioning, the query can be applied to each fragment giving a completed result set at each leaf: the union operator can then be applied to such complete results though any internal nodes to the top nodes. Only one result set is transmitted from each node in the topology. This has significant cost implications for different types of partitioning methods when Boolean searches are used. In both types of partitioning methods, the client is given the full result set.

5.2 Proximity Operations

All operations are complex queries in TermId partitioning, but simple query processing can be used as per Boolean operation for DocId partitioning. The distribution and collection of results is identical to Boolean operations processing. All merges are done using the document identifier. In both types of partitioning methods, the client is given the full result set.

5.3 Term Weighting Operations

Some of these weighting functions require document length information: we retrieve this information from the document map as necessary. Another very important aspect of term weighting is the issue of collection statistics with respect to partitioning method. If we treat fragments of the Inverted file as being in a global database we need to exchange data between the fragments if required. For example in term frequency we need use add all occurrences of a term from all fragments when using document id partitioning: such addition is not necessary when using term id partitioning since the term frequency is available in one fragment. We therefore need to adjust our term weighting operations to suit the partitioning method being used. Other statistics such as average document length and total number of documents in the collection are also affected. It should be noted that term weighting is possible on independent collections [12,13], but the discussion of this subject is outside the scope of our research: our aim is too address the issue of term statistics across fragmented Inverted files. Once weights for each element of the set have been generated we can then use a PLUS set weight operation which is again a special case of union: if two document id's are in both sets we add their weights otherwise we insert the unchanged posting record in the result set. Again different types of operations are required in differing partitioned methods. Using document id partitioning we can sort local results and only send the top n documents from that build to the parent node. With term id partitioning we cannot sort the results set until all data has been received from all fragments. This would appear to give document id partitioning methods an advantage over term id in that less communication is needed, but more sorts are required in document id partitioning. The Weighted Query method is similar to Simple Query operation but differs in that a sort is needed on the final result set. Only the n of the top set documents identified by the weighting operation are presented to the user.

6. RELEVANCE FEEDBACK ON PARTITIONED INVERTED FILES

We support relevance feedback on partitioned Inverted files for query expansion. The method we use is based on assigning a weight for a term called the Term Selection Value derived by Robertson [14]. There are three phases to the query expansion method we use: term extraction, assigning a Term Selection Value to each term and choosing the terms which constitute the reformed query.

6.1 Term Extraction

On inspection of documents which have been presented to the user from his or her initial query, a number of documents are marked as being relevant to a users information need. The number of

chosen documents is assigned to R - the number of know relevant documents for a request. An index is created for this document sent in the same way as described in [15], by taking each document analysing it and adding its information to the index. We are therefore reusing code used in the Indexing module. A porter stemming method is applied to the documents, and a list of stop words defined by Fox [16] are not indexed. When the indexing has been completed each element of the index is inserted into the queries candidate set ready for the next stage of processing.

6.2 Term Selection Value Calculation

In order calculate the Term Selection Value the terms in the candidate set are broadcast to each fragment, and if terms are dealt with in that fragment the no of postings for that term is recorded. In term id partitioning this means a simple assignment of a value (or set merges and an addition if load and stem operation is used). However the requirements for document id partitioning are more complex in that term frequencies of each term from the separate fragments must be added together (on top of the requirements set by term id partitioning). Once this collection information has been calculated we can apply the Term Selection Value function to each element of the candidate set [4,14], using the evidence gathered from relevance feedback in the generated Index for relevance documents.

Note: declare TSV weight here?

6.3 Select Terms for the Revised Query

The last stage is to select the best set of terms (10 or 20 is suggested as being reasonable numbers [4]) and insert them in the current term set which is then applied to the Inverted file aiming get a more relevant set of documents for presentation to the user. The original set of current terms is removed (but still exist in the candidate set). If the user wants to start again, both current term and candidate term sets are cleared.

7. HARDWARE USED

PLIERS is designed to run on several parallel architectures and is currently implemented on those which use Sun Sparc and DEC Alpha processors. All results presented in this paper were obtained on an 8 node Alpha farm and 8 nodes of a 12 node AP3000 at the Australian National University, Canberra. Each node has its own local disk: the Shared Nothing Architecture [3] is used by PLIERS. For the Alpha farm, each node is a series 600 266Mhz Digital Alpha workstation with 128 Mbytes of memory running the Digital UNIX 4.0b operating system. One of the nodes has a RAID disk array attached to it and other nodes can access the RAID using NFS. Two types of network interconnects were used: a 155 Mbytes/s ATM LAN with a Digital GIGASwitch and a 10 Mb/s Ethernet LAN. Search requests were submitted on both types of Networks, but indexing was only done on ATM. The Fujistsu AP3000 is a Distributed Memory Parallel Computer using Ultra 1 processors running Solaris 2.5.1. Each node as a speed of 167Mhz. The machine we used has 12 nodes, but only 8 are available on a partition. The torus network has a top bandwidth of 200 Mbytes/s per second.

8. DATA AND SETTINGS USED

The data used in the experiments was the BASE1 and BASE10 collections, both sub-sets of the official 100 Gigabyte VLC2 collection [14]. The BASE1 is 1 Gigabytes in size, while BASE10 is approximately 10 Gigabytes in size. For the distributed build method we use the BASE1 collection only creating indexes on 1 to 7 processors and searches initiated on all of those indexes. The BASE1 and BASE10 collections were used for the local build method, running queries on 8 nodes down to 1 node. The queries are based on topics 351 to 400 of the TREC-7 ad-hoc track: 50 queries in all. The terms were extracted from TREC-7 topic descriptions using an Okapi query generator utility to produce the final query. The average number of terms per query is 19.58. As stated our atoms size is a paragraph. We examine passage lengths of 5,10,20 and full atom length. The atom step used was always one. For distributed build indexes we do passage retrieval on the top 1000 documents and select the top 20 for evaluation. With local build we do passage retrieval on the top 1000 on each processors and then select the top 20 for evaluation. We use this methodology so comparative analysis can be done with results declared at VLC2 [17].

- who and how are queries for boolean and adj ops formed? 1) me probably! 2) manually!

9. SEARCH RESULTS FROM DOCID PARTITIONING

- include retrieval effectiveness results
- ap1000 docid only
- network version docid, termid
- what topologies are used? may just leave out internal nodes (do a test?)
- short discussion of effectiveness results

10. SEARCH RESULTS FROM TERMID PARTITIONING

- include retrieval effectiveness results- check should be same as 6 - put it in another section.
- ap1000 docid only
- network version docid, termid
- what topologies are used? may just leave out internal nodes (do a test?)
- short discussion of effectiveness results (should be same as docid cross ref).

! the point behind these experiments is to examine speed not effectivness

11. CONCLUSION

- to be done later
-docid vs termid search speed

12. Acknowledgements

This work is supported by the British Academy under grant number IS96/4203. We are also grateful to ACSys for awarding the first author a visiting student fellowship at the Australian National University in order to complete this research and use of their equipment. We are particularly grateful to David Hawking for making the arrangements for the visit to the ANU.

References

[1] S. E. Robertson, Overview of the OKAPI projects, Journal of Documentation, Vol 53, No 1, January 1997.

[2] JEONG, B., *and* OMIECINSKI, E., Inverted file partitioning schemes in multiple disk systems, *IEEE Transactions on Parallel and Distributed Systems*, *6* (2), 1995, 142-153.

[3] A. MacFarlane, S.E.Robertson, and J.A.McCann, Parallel Computing in Information Retrieval - An updated Review, Journal of Documentation, Vol. 53, No. 3, June 1997.

[4] S.E. Robertson and K. Sparck Jones, Simple, Proven approaches to Text Retrieval, University of Cambridge Computer Laboratory, Technical Report No. 356.

[5] S.Jones, T. Do, M Hancock-Beaulieu, A.Payne and S. Robertson, Query Modelling for IR Interface Design, In: F. Johnson, (ed), Proceedings of the 17th Colloquim of the British Computer Society Information Retrieval Specialist Group, Crewe, 1995, The New Review of Document and Text Management, No 1 1995.

[6] S. Wartik, Boolean Operations, In: W. Frakes and R. Baeza-Yates (Eds), Information Retrieval: Data Structures and Algorithms, Prentice-Hall, 1992.

[7] S.E.Robertson and S. Walker, On the logic of search sets and non-Boolean retrieval, Unpublished paper. City University.

[8] A. MacFarlane, S.E.Robertson and J.A.McCann, On Concurrency Control for Inverted Files, Proceedings of BCS Colloq. Manchester 1996.

[9] S.E.Robertson, and K. Sparck Jones, Relevance Weighting of Search Terms, Journal of the American Society for Information Science, May-June 1996.

[10] S.E.Robertson, S.Walker and M.M.Hancock-Beaulieu, Large test collection experiments on an operational, Interactive systems: Okapi at TREC, Information Processing and Management, Vol. 31, No.3. 1995, 345-360.

[11] S.E. Robertson, S. Walker, S. Jones, M.M. Beaulieu M. Gatford and A. Payne, Okapi at TREC-4, In: D.K.Harman, ed, *Proceedings of the Fourth Text Retrieval Conference, Gaithersburg, U.S.A, November 1995*, Gaithersburg: NIST 1996.

[12] A. Singhal, AT&T at TREC-6, In: D.K.Harman, ed, *Proceedings of the Sixth Text Retrieval Conference, Gaithersburg, U.S.A, November 1997*, Gaithersburg: NIST (too appear).

[13] G.V.Cormack, C.L.A.Clarke, C.R.Palmer, and S.S.L.To, Passage-Based Refinement (MultiText Experiments for TREC-6), In: D.K.Harman, ed, *Proceedings of the Sixth Text Retrieval Conference, Gaithersburg, U.S.A, November 1997*, Gaithersburg: NIST (too appear).

[14] S.E.Robertson, On Term Selection for Query Expansion, Documentation Note, Journal of Documentation Vol 46, No 4, December 1990.

[15] A.MacFarlane, S.E.Robertson and J.A.McCann, Parallel methods for the generation of partitioned Inverted files, In preparation.

[16] C. Fox, A stop list for General Text, SIGIR FORUM, ACM Press, Vol 24, No 4, December 1990.

[17] D. Hawking, N. Craswell and P. Thistlewaite, Overview of TREC-7 Very Large Collection Track, In: D.K.Harman, ed, *Proceedings of the Seventh Text Retrieval Conference, Gaithersburg, U.S.A, November 1998*, Gaithersburg: to appear.

[18] TOMASIC, A., *and* GARCIA-MOLINA, H. *Performance of Inverted Indices in Shared-Nothing Distributed Text Document Information Retrieval Systems*. Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems, IEEE Computer Society Press: Los Alomitos (CA),

[19] George Kingsley Zipf, Human Behavior and the Principle of Least Effort, Addison-Wesley Press, Cambridge, Massachusetts, 1949.

[20] N.J.Belkin, R.N.Oddy and H.M.Brooks, ASK for Information Retrieval: Part 1 Background and Theory, Journal of Documentation, Vol. 38, No. 2, June 1982.

[21] S. Kirsch, Infoseek's experiences searching the Internet, SIGIR Forum, Vol 32, No. 2, Fall 1998.