PLIERS AT VLC2

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

ABSTRACT: This paper describes experiments done on the VLC2 collection at TREC-7. Methods used for indexing text is described together with the results: this includes the official collections BASE1, plus some larger unofficial collections named BASE2 and BASE4. Search times on these collections are described and discussed with a particular emphasis on scaleup: for both weighted term search and passage retrieval. The various configurations for experiments are described.

1. INTRODUCTION

This paper is a description of results gained using the PLIERS system on baselines of the VLC2 collection at TREC-7. The research is part of an ongoing effort to study the effects of different partitioning methods for Inverted files in parallel IR systems. In particular we wish to find out which partitioning method yields the best Indexing and Search results with respect to elapsed time as seen by the user. We present the official results for BASE1 only of VLC2, but include some further results from experiments on larger but unofficial collections labelled BASE2 and BASE4, which are approximately 2 and 4 Gigabytes in size respectively. In section 2 we give details of the experiments such as the hardware and software used. The Indexing results are described in section 3 while section 4 describes the results gained on ordinary term weighting search and section 5 describes results gained using passage retrieval search methods. An overview of the results is provided in section 6. A summary is given in section 7.

2. DESCRIPTION OF THE EXPERIMENTS

In this section we describe various aspects of the experiments, such as software and hardware used, description of the data used, query processing details, the retrieval model used in search and measures used in the experiments.

2.1 SOFTWARE USED

PLIERS (ParaLLeL Information rEtrieval Research System) is a parallel text retrieval system currently being developed at City University as part of the first author's PhD studies. The concepts and design used in the development of this software are heavily influenced by the Okapi system [1]. The message passing system namely MPICH [2] is used for data exchange: this system supports the MPI (Message Passing Interface) standard.

2.2 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 at the Australian National University, Canberra. Each node has its own local disk: the Shared Nothing Architecture [3] is used by PLIERS. 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 Mb/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. Figure 1 shows the architecture of the Alpha farm.

2.3 DATA DESCRIPTION

We use a number of collections in our experiments: BASE1, BASE2 and BASE4. BASE1 is an officially defined sample of the 100 Gigabyte VLC2 collection [4] and is 1 Gigabyte in size. The BASE2 and BASE4 collections are subsets of the official BASE10 collection (another sample of VLC2) and were created by varying the number of BASE10 compressed text files put through the indexing mechanism (130 files per node for BASE2 and 260 for BASE4). The BASE2 collection is 2.1 Gigabytes in size, while the BASE4 collection is 4.2 Gigabytes in size.

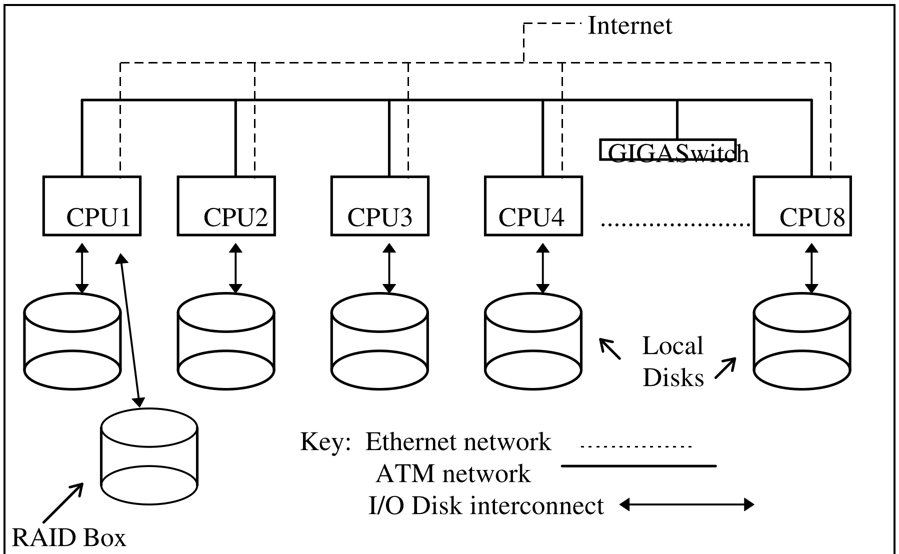


Figure 1 - Alpha Farm Architecture

The strategy used to distribute the BASE1 and BASE10 collections across the nodes was to evenly spread the compressed text file's directories among them as far as possible (an alternative if more time consuming strategy is to do it by file size). The requirement of a distribution strategy is to get the best possible load balance for Indexing as well as Term Weighting and Passage Retrieval search. The distribution was done before the Indexing program was started, and is not included in the timings.

Inverted Files were used with a Dictionary File that contains: keywords, the number of occurrences in the collection of that keyword and a pointer to an Inverted list in the Postings File. Each element of the Postings file contains: a posting with a document identifier, the number of occurrences of a word in that document, plus a list of positions for a word in that document (if such processing is required). Each position element contains field, paragraph, sentence and word position data together with the number of preceding stop words. Only the paragraph data was used in these experiments (for passage retrieval). A Document map file was used to record document length and its start location in the text file as well as the document identifier.

Two types of Inverted files were used for experiments: one type that recorded position information (necessary for passage retrieval and adjacency operations if required) and one that recorded postings only in the Inverted list.

2.4 INDEXING METHODOLOGY

The strategy used for indexing was a parallel one using the document identifier partitioning method, keeping all indexing builds local to the processor. We define document identifier partitioning as Inverted file data that is distributed over a number of partitions, each partition having a unique set of document identifiers [5]. We name the method used for parallel indexing with its data distribution strategy "Local Build Document Identifier Indexing". The method reduces communication to the minimum: after the initial distribution of documents, only control messages are sent. Fragments of the Inverted file are distributed across local disks. Each fragment of the whole Index is an Index of the sub-collection held on that node. We do not keep a super dictionary, which would record all keyword and occurrence data in one central location. Figure 2 shows the Indexer topology.

For each local Index build we used a stop word list of 450 words to filter out unwanted terms and was supplied by Fox [6]. HTML tags were stripped from the text and ignored if not used for specific reasons such as identifying paragraphs <p> and the end of document </DOC>. Each identified word was put through a Lovins stemmer, supplied by the University of Melbourne, and indexed in stem form. Numbers were not indexed. A large amount of in-core memory is pre-allocated in blocks by each indexing process, and documents are analysed until one of several criteria is reached: run out of keyword block, posting block or position block space. When one of the criteria is satisfied, the current analysis is dumped to disk as an intermediate Index, so that the in-core memory can be used for the next set of documents. When all documents have been analysed, the intermediate Indexes are merged together to create the final Index and deleted. We attempted to keep keywords in main memory for the duration of the indexing, but found that we ran out of keyword space fairly quickly on the data sizes we used. Therefore intermediate dictionary files had to be created as well.

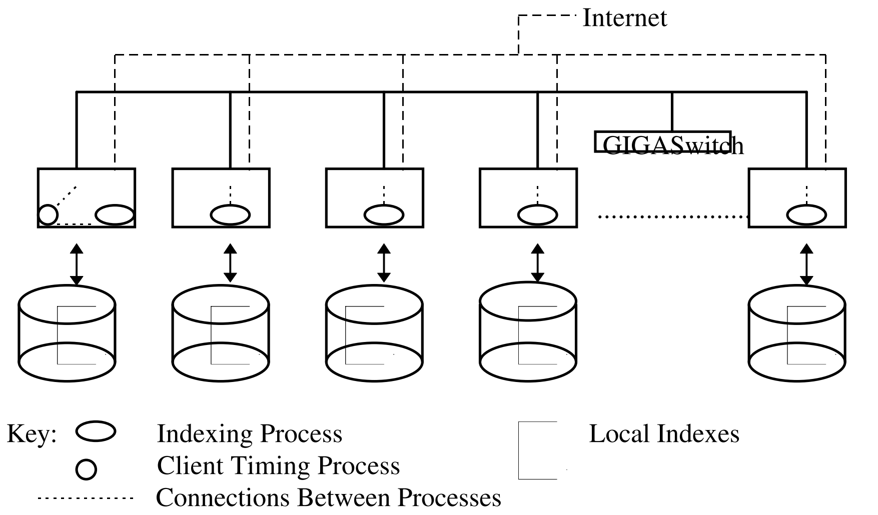


Figure 2 - Indexing Topology

2.5 QUERY PROCESSING

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 and put through the Lovins stemmer to produce the final query. The average number of terms per query is 19.58. Queries were processed over the full collections and then subsets (reducing in 1/8th steps to 1/8th of the collections). All queries where submitted sequentially to the system, distributing the queries to all leaf nodes in the system and processing the individual queries in parallel (see figure 3). The batch query client process reads in queries and sends them to the Top Search process. The top search process then sends the query to all leaf processes, awaiting results. The results come back in the inverse direction of the Query. We used three different cases of query processing: term weighting search in the presence and absence of position data in the Inverted file, and Passage Retrieval.

The passage retrieval techniques used here was implemented in Okapi at TREC-3 [7], and is done on arbitrary passages, iterating through contiguous sequences of text atoms in order to find the best weighted passage. The text atoms used were paragraphs. There are two stages in the process. An ordinary term weighting search including a sort identifies the top 1000 top ranked documents in the first stage. The second stage applies the passage retrieval algorithm to the top ranked documents from the first stage. With our method we apply the passage retrieval algorithm to 1000 documents per node, therefore passage retrieval is done on 8000 documents for the full collections, 7000 for 7/8th down to 1000 for 1/8th.

2.6 RETRIEVAL MODEL

The model used for search is the Robertson/Spark Jones Probabilistic model [8] and all Searches, including the ordinary term weighting operation and passage retrieval, use the BM25 weighting function [9]. The tunning constant settings were 2.0 for k1 and 0.6 for B.

2.7 MEASURES USED

For the official results we supply average elapsed times as seen by the user for all query processing cases in search and the total elapsed time for Indexing. For search we give the average elapsed time on the ATM network only as the differences in time with Ethernet elapsed times were found to be small. All search experiments on top of the official submitted results were aimed at measuring scaleup on the chosen method of data partitioning for Inverted files: namely document identifier partitioning. Scaleup is defined as the ability to process a large job in the same time as a smaller job [3]. The ideal scaleup would be 1.0, and any figure higher than this gives the loss due to distribution. Scaleup was measured by obtaining results on the on full BASEx collection and then working backwards getting 7/8th of BASEx to 1/8th of BASEx and comparing the search results on all fragments of the collection. For each query processing case we supply the scaleup from 1 to 8 leaf processes (average elapsed time on 8 leaf processes divided by average time 1 on leaf process) and comment on the scaleup line (see appendix 1). For Indexing we use a measure of load imbalance LI, defined as being the ratio of the maximum elapsed time divided by the average elapsed time: a perfect load balance would achieve an LI of 1.0 [10].

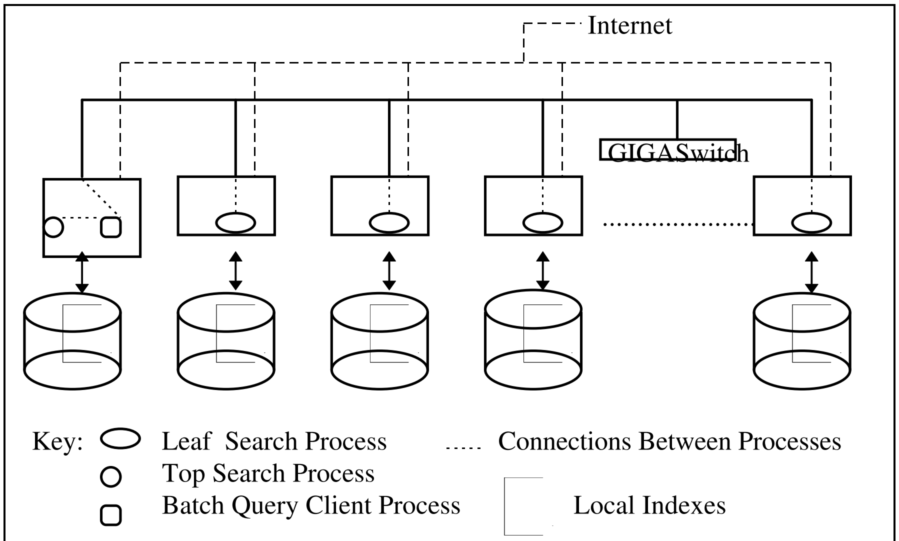


Figure 3 - Query processing topology for search

Inverted file sizes and the overhead they incur with respect to the text are declared. We measure the costs of having position information in the Inverted file for searches that do not require such: this is the query processing case of term weighting search in the presence of position data. It should be noted that document identifier partitioning necessarily incurs a space overhead on the dictionary file as keywords may be on several or all nodes: we do not measure this overhead.

3. INDEXING RESULTS

In this section we give timings for Index building for the BASE1, BASE2 and BASE4 collections and the resultant sizes of the Inverted files for these collections.

3.1 OFFICIAL BASE1 RESULTS

The time taken to index BASE1 was 416 seconds with position data and 286 seconds without position data. The Inverted file with position data was 334 Megabytes in size, while the Inverted file with postings only was 127 Megabytes in size: 35% and 13% of the text respectively. The cost of recording position data in the files was therefore an extra 130 seconds in time and an extra 22% of the text for disk space. The LI for indexing with position data was 1.11, while 1.07 was recorded for indexing without position data: both reasonable measures for load imbalance.

3.2 BASE2 RESULTS

The time to index BASE2 was 5867 seconds with position data and 589 seconds without position data. The Inverted file with position data was 620 Megabytes in size, while the Inverted file with postings only was 275 Megabytes in size. The Inverted file with position data was 29% of the text, while the Inverted file with postings only was 13% of the text. While the extra costs for file space with position data were acceptable for the Inverted files, the extra time needed were not acceptable being nearly 10 times of that required for indexing with postings only. The LI figures reflect this mismatch: an LI of 3.44 was recorded for indexing with position data, while a figure of 1.06 was found for postings only. The reason for this mismatch is discussed in section 3.4.

3.3 BASE4 RESULTS

The results on BASE4 were done on Inversion without position data because of problems with the merging stage of the Indexing: this prevented us from going any further with the experiments (see section 3.4). The time to index the BASE4 collection without position data was 10,044 seconds, with a very poor LI of 3.52: the reason for this is discussed in section 3.4. The Inverted file was 557 Megabytes in size: therefore the Inverted file was 13% of the text file.

3.4 COMPARISON OF INDEXING RESULTS

The times for indexing on BASE1 and BASE2 with no position data were found to be acceptable as was the LI measure, recording figures of just over 1. However the figures for BASE2 with position data and BASE4 with postings only did not meet the required performance. The reason for this is twofold: the merging method used did not scale as times deteriorated with data size and the use of recursion in list processing caused problems by using up the stack space. Elapsed indexing times gradually became worse with the amount of memory needed to process the data and no further progress could be made. The method was unusable beyond BASE4 with postings only data. The official collection BASE10 was not dealt with because of the problems (the full VLC would not have been done because of restrictions on resources). Work has been completed which replaces the merging phase of the indexing in order to make it both scaleable and be able to handle much larger data sets. Iterative techniques for list processing have been introduced to overcome the stack space problem.

Of all the measurements taken with respect to indexing, the Inverted file size cost was found to be good, being around 29/35% when position data is recorded and 13% when postings only are recorded.

4. TERM WEIGHTING SEARCH RESULTS

We supply the results for term weighting search in this section including the retrieval efficiency and effectiveness for the BASE1 collection and retrieval efficiency results for BASE2 and BASE4 collections.

4.1 OFFICIAL BASE1 RESULTS

A. Timings

The average elapsed search time for 50 queries on the full BASE1 collection operating on Inverted files with position data was 1.74 seconds. The scaleup from 1 to 8 leaf processes was found to be 2.6. The data for queries on an Index with postings only delivered a much better average response time with 0.648 seconds. The scaleup curves for Inversion with position data were generally encouraging apart from a sharp rise in elapsed average time from 7 to 8 leaf processes: the same is found on Inversion without position data (see appendix 1, BASE1 timings).

B. Retrieval effectiveness results

The id of the official term weighting run was pliers1-vlc2. The results for this run were very disappointing with only 80 relevant documents retrieved and a precision of 0.08. On investigation it was found that the problems discovered in Indexing caused data corruption in the postings lists. The removal of this data corruption problem improved the precision to 0.126 with the original tuning constant settings (for k1 and B) and this was increased to 0.132 with k1 set as 1.0. This represents an increase of 65% over the original results. A further evaluation on the top 1000 documents on the original tuning constant settings found 386 relevant retrieved documents so there is still much room for improvement, i.e. there are more relevant documents in BASE1.

4.2 BASE2 RESULTS

The average search elapsed time for 50 queries on the full BASE2 collection operating on Inverted files with position data was 3.14 seconds on the ATM network. The scaleup from 1 to 8 leaf processes was 2.04. For searches in the absence of position data the search times on the 50 queries were much better with average elapsed times of 1.11 seconds: scaleup from 1 to 8 leaf processes was 2.92. The scaleup curves for Inversion with position data were generally encouraging apart from a sharp rise in elapsed average time from 7 to 8 leaf processes: the same is found on Inversion without position data (See appendix 2, BASE2 timings).

4.3 BASE4 RESULTS

The search average time for 50 queries on the full BASE4 collection operating on Inverted files with postings only data was 3.14 seconds. Scaleup from 1 to 8 leaf processes was 2.357. The scaleup curves are identical, but there is a difference on 6 leaf processes with a higher figure for the Ethernet results. There is a sharp rise in elapsed average time again from 7 to 8 leaf processes (BASE4 timings, appendix 3).

4.4 COMPARISON OF TERM WEIGHTING SEARCH RESULTS

The average results for the 50 queries on all data sets are in the main good for all types of Inverted file type and network type. Comparing searches on Inverted files with position data we found that the average elapsed time for 50 queries on BASE2 where slightly less than double than those on BASE1. The comparison with postings data only was less favourable: 50 queries on BASE2 was again just over double that of BASE1, but the queries on BASE4 took just under three times longer than on BASE2 and just over five times longer on BASE1. This is clearly a problem with the scalability of the sequential search algorithm. We put this down to a merging failure in the indexing process, since the method used did not guarantee a unique keyword per fragment: therefore increasing the amount of I/O for any given query term. Comparing the overhead in search speed on queries that do not require position data we find that at BASE1/BASE2 search times for searches on Inverted files were 2.7/2.8 times longer than the same queries on files with postings data only. The scaleup curves for both types of networks appear to be generally favourable on all types of Inverted configurations, apart from a sharp rise in elapsed average time on 7 to 8 leaf processes.

5. PASSAGE RETRIEVAL RESULTS

We supply the results for passage retrieval search in this section including the retrieval efficiency for BASE1, BASE2 and BASE4 collections as well as the retrieval effectiveness results for BASE1.

5.1 OFFICIAL BASE1 RESULTS

A. Timings

The average search time for 50 queries on the full BASE1 collection was 3.49 seconds. The scaleup from 1 to 8 leaf processes was found to be 2.09, much better figures than for term weighting search: however the average search times are just under double that of term weighting search. The scaleup curves are broadly encouraging apart from a sharp rise in elapsed average time from 7 to 8 leaf processes (see appendix 1).

B. Retrieval effectiveness results

The id of the official passage retrieval run on BASE1 was pliers2-vlc2. The data corruption problem discussed in section 4.1 above also affected passage retrieval with a precision of 0.056 on the official run with only 56 relevant retrieved. The revised program produced 79 relevant retrieved documents with a precision of 0.079 but the results are still a long way behind the revised term weighting results declared above. The best revised results were found with k1 set as 0.5 yielding a precision of 0.091 (91 relevant retrieved documents). A further evaluation on the top 1000 documents on the original tuning constant settings found 339 relevant retrieved documents. Significant further investigation is merited into why passage retrieval is not performing as well as hoped and giving improved retrieval effectiveness over term weighting retrieval.

5.2 BASE2 RESULTS

The average search time for 50 queries on the full BASE2 collection was 6.17, just under double the time for the average elapsed search time on passage retrieval over the BASE1 collection. The scaleup from 1 to 8 leaf processes was found to be very good indeed with figures of 1.56: by far the best scaleup recorded in these experiments. There is something of a strange effect in the scaleup curve going from 3 to 4 leaf processes in that elapsed time on 4 leaf processes is better than on 3. The effect is found on both ATM and Ethernet networks. The 4 node search is a superset of the 3 node search. For the moment we are at a loss to explain this and the problem is currently under investigation. There is a sharp rise in elapsed average time from 7 to 8 leaf processes on the scaleup curve again (see appendix 2).

5.3 COMPARISON OF PASSAGE RETRIEVAL RESULTS

With respect to passage retrieval for the 50 queries over BASE1 and BASE2, it was found that the average elapsed search time for BASE2 were 77% larger than BASE1 (6.17 seconds for BASE1 compared with 3.49 seconds for BASE2). This is encouraging as BASE2 is just over double the size of BASE1. However, the average results for passage retrieval are expensive compared with term weighting search, with searches on average being slightly less than double with term weighting searches on Indexes with position data and a factor of just over five times that of term weighting searches on Indexes with postings only. The method therefore would only be worthwhile if gains could be obtained in retrieval effectiveness.

6. OVERVIEW OF RESULTS

We draw the threads of both indexing and search together in this section, discussing various aspects found in the experiments including the issues of Load Imbalance, scalability, communications costs, process mapping and a discussion of PLIERS as implemented.

6.1 LOAD IMBALANCE

DATABASE	**TEXT**	**INDEX**		**TIME**	
		Posting Only	*Position Data*	*Posting Only*	*Position Data*
BASE1	1.05	1.09	1.09	1.07	1.11
BASE2	1.02	1.07	1.19	1.06	3.44
BASE4	1.03	1.05	-	3.52	-

Table 1 - Comparison in LI between Times, Full Text and Indexes

From Table 1 it is clear than in the most part, there is not that much different between the LI for actual text size, build time and Index file sizes. The anomalies are found in BASE2 with position data and BASE4 with postings only and these are caused by the list processing problem described below.

6.2 SCALABILITY ISSUES

An important question to ask is how far the PLIERS software would scale to systems that use more than 8 nodes say up to 1000 nodes. With Indexing we feel confident that the method would scale to systems with much larger numbers of processors. The scalability of the Search process is perhaps not so clear cut because of the communication involved which must eventually affect its performance (see section 6.3). However, the Search process should scale to systems beyond 8 leaf processes (see section 6.4 with respect to the sharp rise in elapsed average time from 7 to 8 leaf processes). These statements are made in the knowledge that the scalability of the sequential algorithms utilised leaves something to be desired (see section 6.5 below).

6.3 COMMUNICATION COSTS

Communication costs are kept to a minimum in the techniques described in this paper. With Indexing only a small number of control messages are used and the Indexers can proceed independently without the need for any communication between each other. With Search, only queries and answers are transmitted rather than whole data sets slowing the computation. Since average elapsed times on both Ethernet and ATM networks are similar we have strong evidence that reducing communication as far as possible in parallel IR systems are important. Some extra communication in search is required in order to satisfy the requirements of the term weighting model: as a keyword may reside on several fragments we need to accumulate the term frequency so that accurate term weighting can be done. We treat the fragmented collection has a whole collection, rather than a number of separate sub-collections. The extra communication is the price to be paid for not keeping a super dictionary, recording the term frequency in a central location. Such a technique would reduce communication further, but would require extra storage space.

6.4 PROCESS MAPPING

One of the most important aspects of Parallel Computing is the mapping of processes to processors: what processor should go were and how many processes should we allow on one processor. We wish to ensure that processors are not overloaded, skewing the computation and reducing the load balance. In most of our search experiments we were able to place the query batch client and top search node on the same processor, but on a different one from the leaves. This was not the case when all 8 nodes for needed for processing, therefore the top search node and batch query client had to be placed on the same processor as a leaf node. These three nodes on one processor competed for CPU and memory resources. This explains in part the sharp rise in elapsed average time from 7 to 8 leaf processes found in all searches on the full collections. It is possible to get by this in MPICH by using a processor of a different type for the query batch client and top search node. This however does require some reprogramming to deal with differences in fundamental types and representations for data. Not all MPI implementations support this facility, however.

6.5 IMPLEMENTATION ISSUES

As can be seen, particularly with some of the indexing results, there is evidence of non-linearity in system behaviour with respect to elapsed time. As discussed earlier it was found that the merging technique used, by not guaranteeing a unique keyword for the dictionary file for any inverted file fragment caused problems in both Indexing and Search. With indexing it required more write I/O's when saving data (1 I/O extra per keyword duplicate in the dictionary file, plus one for the Inverted list), and with growing data sizes the likelihood of duplicates increased thereby increasing times. These extra duplicates impacted on Search at query time by imposing an extra I/O per duplicate in order to read in the inverted lists for a term. A new merging technique has been implemented and is currently being tested which will eliminate the non-linearities found in the experiments. The use of recursion in list processing was found to be problematic requiring the use of stack space in vast quantities. No matter how much stack space is made available, using such a technique would eventually run into problems with increase in data size. The use of iterative techniques in for list processing is therefore not recommended for IR systems that handle very large data sets. This has been a hard if worthwhile lesson to be learned for the first author. Iterative techniques for list processing have now been implemented.

7. SUMMARY

Overall the results presented in this paper are encouraging, but it is important to recognise that further work is needed to improve the performance of individual nodes of both Indexing and Search (both term weighting search and passage retrieval). Indexing speed is very good on BASE1 and BASE2 with no position data, and good load balance figures (LI) are recorded, all just over the 1 mark. It is quite clear from the data that the method used provides useful scaleup on searching with all scaleup curves being very near flat. The ATM and Ethernet curves on search are almost identical on all collections, and we confirm that using the Shared Nothing architecture transmitting results only rather than whole data sets is useful in parallel IR systems. We also infer that the method defined as "Local Build Document Identifier Indexing" is a useful method and data distribution strategy for fragmented Inverted files in parallel IR systems. Clearly retrieval effectiveness could also be improved further, and that significant further investigation is merited into the difference in results between term weighting search and passage retrieval.

8. ACKNOWLEDGEMENTS

This research is funded in part by British Academy Grant No 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, in particular to David Hawking for making the arrangements for the visit.

REFERENCES

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

[2] Gropp, W., and Lusk, E., Users guide for MPICH, a portable Implementation of MPI, Mathematics and Computer Science Division, Argonne National Laboratory, University of Chicago, 6 July 1998.

[3] DeWitt, D., *and* Gray. J. Parallel database systems: the future of high performance database systems. *Communications of the ACM*, *35* (6), 1992, 85-98.

[4] VLC Web Page URL http://pastime.anu.edu.au/TAR/vlc.html

[5] 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.

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

[7] Robertson, S.E., Walker, S., Jones, S., Hancock-Beaulieu, M.M. and Gatford, M., Okapi at TREC-3, In: Harman, D.K., *Proceedings of Third Text Retrieval Conference, Gaithersburg, USA, November 1994*.

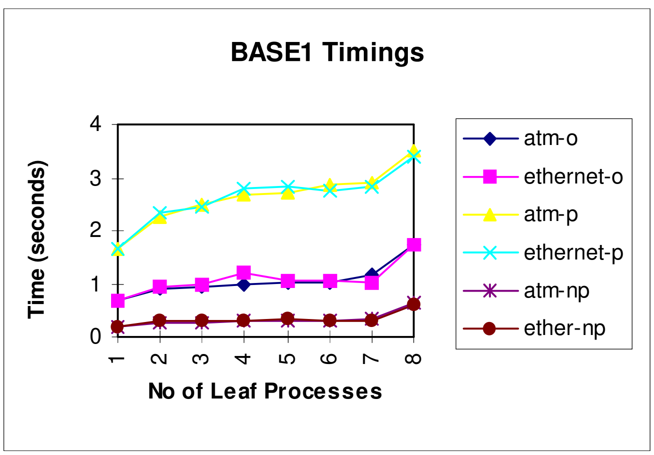
[8] S.E.Robertson, and K. Sparck Jones, Relevance Weighting of Search Terms, Journal of the American Society for Information Science, May-June 1996.

[9] 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.

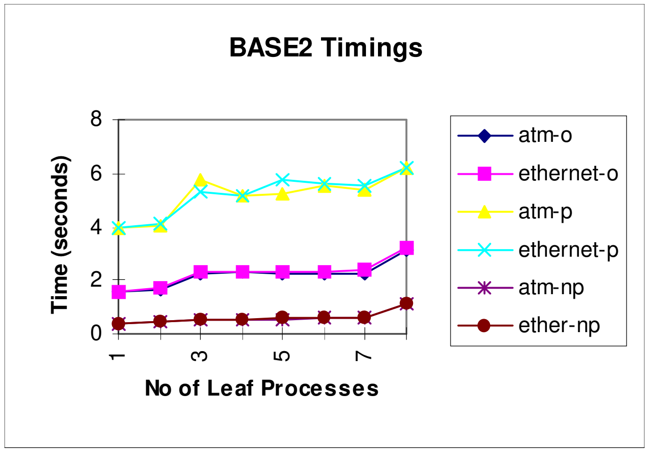
[10] Hawking, D. *The Design And Implementation Of A Parallel Document Retrieval Engine*. Technical Report TR-CS-95-08, Department of Computer Science. Canberra: Australian National University, 1995.

APPENDICES

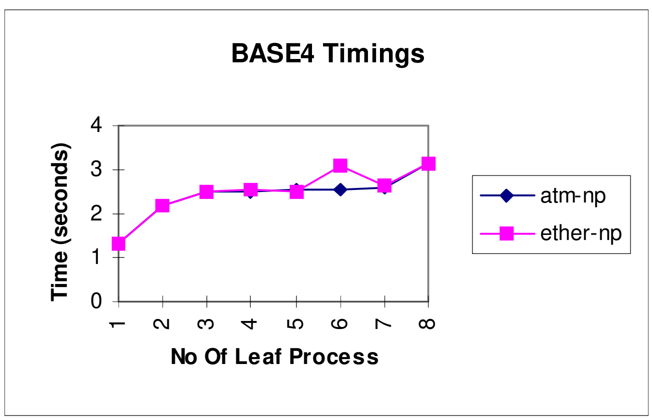
Note on Appendices: Each line on the graph is labelled with either the type of network e.g. ATM or Ethernet together with the type of query processing used: o for term weighing on Inverted files with position data, p for passage retrieval and np for term weighting on Inverted files with postings data only.



Appendix 1- Scaleup for Search on BASE1



Appendix 2 - Scaleup of Search on BASE2



Appendix 3 - Scaleup of Search on BASE4