Parallel Computing for Passage Retrieval

A. MacFarlane[1], S.E.Robertson[1,2] and J.A.McCann[3]

[1]Centre for Interactive Systems Research, City University, London
[2]Microsoft Research Ltd, Cambridge
[3]Department of Computing, Imperial College London

{email: andym@soi.city.ac.uk}

Abstract: In this paper we examine methods for both speeding up passage processing and examining more passages using parallel computers. We vary the number of passages processed in order to examine the effect on retrieval effectiveness and efficiency. The particular algorithm we apply has previously been used to good effect in Okapi experiments at TREC. We describe this algorithm and our mechanism for applying parallel computing to speed up the processing.

1. Introduction

This paper addresses the issue of applying parallel techniques to a computationally intensive passage retrieval method used by Okapi at the Text Retrieval Conference (TREC). This method of passage retrieval would require vast CPU resources; therefore parallel techniques could be useful to speed up the process. The research presented here is part of an overall study of parallelism and data distribution methods in information retrieval [1]. Our research has a two fold purpose: to show a reduction for the elapsed time of this passage retrieval algorithm and examine more of the search space to obtain increased retrieval effectiveness.

The aims and objectives of our research are set out in section 2. We define and describe passage retrieval in section 3, then outline the Okapi algorithm in section 4. We then describe our work. The implementation of the parallel algorithm is described in section 5. The hardware and software used is declared in section 6. The data and settings used for experiments are described in section 7. We discuss retrieval efficiency and effectiveness results together in section 8. A conclusion is given in section 9.

2. Experimental aims and objectives

The aim of the research presented in this paper is two fold: is it possible to decrease the run time costs for the passage retrieval algorithm using parallelism and; is it possible to increase the number of relevant documents retrieved by examining more of the search space. We therefore have a two-fold hypothesis to examine in this paper:

Hypothesis 1: Using parallel computing will increase the retrieval efficiency of passage processing, in terms of faster processing times (speedup) or examining more documents in the same time (scalability).

Hypothesis 2: Examining more passages will increase the retrieval effectiveness of query processing, that is more relevant documents will be found.

We vary the number of documents processed in order to examine the validity of these hypotheses. In terms of retrieval efficiency this allows us to examine different rates of passage processing, while for effectiveness we can show how the retrieval of relevant documents varies if at all.

3 Description of passage retrieval

The work that motivates the research described in this paper is from Okapi experiments conducted within the TREC conference framework in particular Okapi at TREC-3 [2]. We describe the Okapi algorithm used in section 4, but first we give an overview of what passage retrieval actually is and the context in which we use it. Passage retrieval can take on a number of forms and we list some examples here:

- Retrieval of part of a document that is most likely to be of interest to a user.
- To help in the identification of relevant documents.
- A component of a question-answering system.
- A component of a query-specific summarising system.

Our interest is in the second of these and all of the discussion that follows concentrates on using passage processing for the identification of more relevant documents. It should be noted that a passage could be a whole document.

Retrieval is done on the basis of text atoms, which is the smallest piece of text manipulated by passage processing. Atoms can be paragraphs [2], blocks, sentences, words or even characters: we use an atom size of five sentences. A passage is usually defined as a contiguous sequence of atoms, and new passages may be generated iteratively from old ones by adding or removing blocks of atoms in increments: we use an increment of one. Once the text atom and its incremental level are defined we can either define static or arbitrary lengths for passages based on the atom and the increment size. A fixed length passage starts from a given atom position, and the passage weight is computed using n atoms. An arbitrary length passage mechanism relaxes the fixed length constraint and allows computation of passage weight for a passage of arbitrary length. This is rather expensive computationally requiring $O(a^3)$ steps [2] where a is the number of atoms processed. We compromise between the two methods and have maximum length passages, reducing the time complexity to $O(a(a-1)/2)$: this is the complexity of the algorithm we use in this paper. A further refinement is the choice of increment. We may disallow overlapping passages to reduce the computation even further, but we do compute on overlapping passages. We may consider single best passages or combining evidence from several passages: we use the former. Passages and atoms are determined at index time in our methods: we save this data in the inverted file when the index is generated.

4. The Sequential Algorithm

The basic idea behind the passage retrieval method in Okapi is to iterate through contiguous sequences of text atoms and find the combination that yields the best weight for that document. This procedure is done on inverted files. The algorithm is shown in figure 1.

```
Function do_passage( IN: document data ) RETURN OUT: weight

  For( start=0; start < no_of_atoms; start++ )
     For( finish=start+INCREMENTAL_VALUE;
          finish< no_of_atoms AND finish-start < MAX_PASSASGE_LEN;
          finish=finish+INCREMENTAL_VALUE )
           If(at least one query term is in start and finish atoms)
                    calculate weight for current passage
           EndIf
           if( current passage weight > largest passage weight )
                    record details of current passage as best passage
           EndIf
     EndFor
  EndFor

  return best passage weight

END do_passage

(1st Phase)
Obtain the top 1000 ranked documents for query terms

(2nd Phase)
Get position lists for the terms in the query.
loop 1000 documents
call do_passage function to obtain best
                        passage for that document
EndLoop
Re-rank the top 1000 documents.
Display top x documents to the user.
```

Figure 1: Algorithm for sequential passage Retrieval

There are two stages to the sequential algorithm: retrieve the top ranked documents (say 1000) and then apply the passage retrieval algorithm to all top ranked documents. Restricting processing to 1000 documents is a time saving process: in principle we could consider all documents. The first phase is a simple probabilistic search as described in [3]. In the second phase a list of positions is obtained for each document and word pair in the query. We do not need to analyse full text since position data is saved in our inverted file during the indexing process [1]. The passage retrieval algorithm is then applied, recording the best passage weight for each document.

The processing requires the weighting of a document for terms in the query sets given the relevant position data, and iterating through the defined passage,

recording the highest weighted passage. In the case of best match functions such as BM25 [4], the requirements may also include the calculation of the passage length. This passage length is available from the inverted file having being recorded at indexing time. Once all documents have been processed the top set is re-ranked and can be presented to the user. We can reduce the time complexity for this method by setting a maximum passage length (the term MAX_PASSAGE_LEN in figure 1), set to twenty atoms in our case [2]. The minimum passage length could be sensibly set to one atom [2]. Our arbitrary length passages therefore range from one atom to a maximum of twenty atoms per passage. A further refinement is to specify the number of increment steps for examined atoms: this can be altered by changing the INCREMENTAL_VALUE constant. It should be noted that if either the start atom or the finish atom contains no query term, we do not calculate the weight for that passage. In general there must be a shorter passage that would be better, given that the BM25 scoring function increases with reducing document length (see appendix 2).

5. The Parallel Algorithm

In this section we describe the parallel implementation of the passage retrieval algorithm described in section 4. There are a number of issues when considering the application of parallelism to the Okapi passage retrieval algorithm. In particular the issue of how the algorithm is applied to the inverted file with a chosen data partitioning method must be addressed. A data partitioning method is a strategy for distributing inverted file data to the nodes in a parallel computer. We define a *node* as being made up of a processor, memory and disk space.

The scheme we use for data partitioning is as follows. Each *node* has its own disk and processor and we split up or partition the inverted file among the disks such that each disk has its own unique set of documents (that is each node has its own sub-collection). Another big issue is how processes in the parallel program can communicate with each other: this network of communicating processes is called a process topology. We use a process topology described in more detail in [3] but we give an example process configuration in appendix 1. We have a number of *leaf nodes* that manages each sub-collection and a *top node* that is an interface between the outside world and the *leaf nodes*. Each node does passage retrieval on a given number of documents say N: we can vary N according to our requirements. All passage processing is done locally on a *leaf node*. Once a *leaf node* has received the query, no further communication is needed until the results have been produced. With each additional *leaf node* we can examine more or less of the search space as required. In this method a total of N*P documents is examined for possible good passages (where P is the number of *leaf nodes*). The approximate time complexity for this method is O((a(a-1)/2) / P): recall that a is the number of text atoms processed.

```
Top node sends query to P leaf nodes
InParallel for P leaf nodes
        (1st Phase)
        Obtain N top ranked documents for the fragments
                query terms.

        (2nd Phase)
        Get position lists for the terms in the query.
        Loop N documents
                call do_passage function to obtain best
                        passage for that document
        EndLoop
        Re-rank the top N documents.
EndParallel
Top node retrieve top x documents from P leaf nodes.
Display x documents to the user.
```

Figure 2: Parallel passage retrieval algorithm

6. Hardware and software used

The system used to do the experiments is the PLIERS system (ParaLLel Information rEtrieval Research System) that has been developed to investigate various aspects of parallelism in IR [1]. All results presented in this paper were obtained on 8 nodes of a 12 node Fujitsu AP3000 at the Australian National University (ANU) in Canberra. The AP3000 is a distributed memory parallel computer using Ultra 1 nodes with clock speed of 167Mhz running Solaris 2.5.1. Each node in the AP3000 architecture has its own local disk: that is a *shared nothing architecture* [1] is used by PLIERS. The torus network has a top bandwidth of 200 Mbytes/s per second.

7. Data and settings used

The data used in the experiments was the BASE1 and BASE10 collections, both sub-sets of the official 100 Gigabytes VLC2 collection for TREC [5]. BASE1 is 1 Gigabyte in size, while BASE10 is approximately 10 Gigabytes in size. We ran queries on 8 nodes for BASE10 and 1 to 8 nodes for BASE1. 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 queries. We used two types of query sets: one based on *title only* (average number of terms per query is 2.46) and one based on the *whole topic* (average number of terms per query is 19.58). The *whole topic* query set has 51 queries, one extra being for VLC2 experiment initialisation [5]. Our timing methodology was as follows: for *title only* queries we declare the average of 10 runs, while we declared the average for 5 runs on *whole topic*. The model we use for term weighting in our experiments is the Robertson/Sparck Jones Probabilistic model [6] and we use the BM25 weighting function (see appendix 2). Recall that the atom size

used in our experiments is 5 sentences using an incremental step of one for atoms. We select the top 20 of all runs for evaluation (the relevance data is targeted at this figure).

8. Experimental results

To facilitate the discussion below (which is the examination of parallel runs) we declare sequential passage retrieval results used for comparison on the BASE1 collection. The average elapsed time in seconds was 1.33 for *title only* queries and 45.6 for *whole topic* queries. The figure for *title only* is acceptable and is within the scope of 10 second response times suggested by Frakes [7]. However *whole topic* response time is not acceptable within such criterion: it does however demonstrate how computationally costly passage retrieval can be. Retrieval effectiveness is increased on all precision points when applying passage retrieval on the BASE1 collection using *title only* queries: precision at 20 was increased from 0.130 to 0.148 (an increase in performance of 13.8%). For all other runs improvements were found only on some of the precision points (see table 1 for effectiveness results on runs without passage processing). It should be noted that we were unable to get the same level of retrieval effectiveness on longer queries as with shorter queries: this merits further investigation.

Collection	**Query Type**	**p@5**	**p@10**	**p@15**	**p@20**
BASE1	*title only*	0.244	0.178	0.149	0.130
	whole topic	0.188	0.172	0.145	0.128
BASE10	*title only*	0.324	0.282	0.273	0.264
	whole topic	0.356	0.298	0.271	0.247

Table 1 - Retrieval effectiveness results (no passage processing)

We use a number of different metrics to examine the retrieval efficiency of parallel passage retrieval methods: average elapsed time in seconds, system throughput, load imbalance or LI, scalability, speedup and parallel efficiency (these metrics are defined in the Glossary). We compare the elapsed times for VLC2 participants [5] with our first set of experiments only (these are the most expensive runs). We use the retrieval effectiveness metric precision (at 5,10,15,20 documents retrieved) as well as passage retrieval statistics such as number of documents and passages processed during a run. We do three sets of experiments using parallelism, varying the amount of passages processed to test our hypotheses. The first is applying passage processing to 1000 documents on each node (we therefore increase the number of documents examined with increasing number of nodes). The next is to apply passage retrieval to 1000 documents overall. Finally we apply the algorithm to reduced document sets of less than 1000 documents.

8.1 Experiment Results on 1000 documents per node

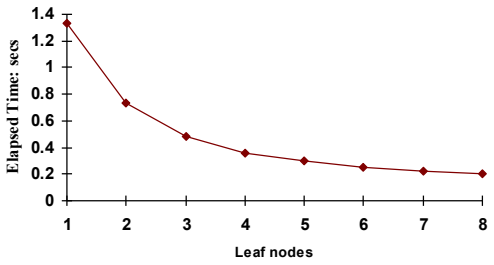


Fig 3. BASE1 [*title only*]: Retrieval efficiency, average elapsed time in seconds for 1000 documents per node

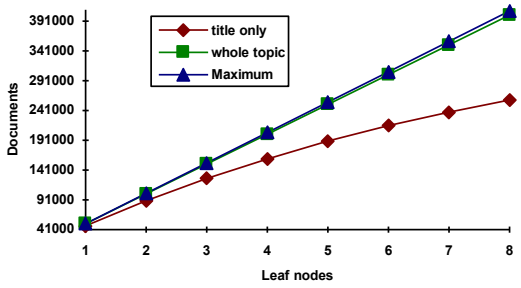


Fig 6. BASE1: Retrieval efficiency, documents processed for 1000 documents per node

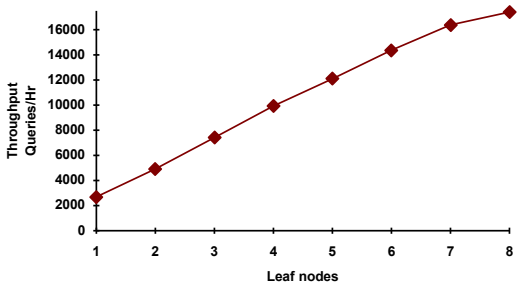


Fig 4. BASE1 [*title only*]: Retrieval efficiency, throughput (queries/Hour) for 1000 documents per node

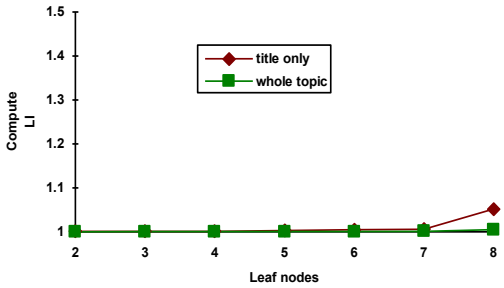


Fig 7. BASE1: Retrieval efficiency, load imbalance for 1000 documents per node

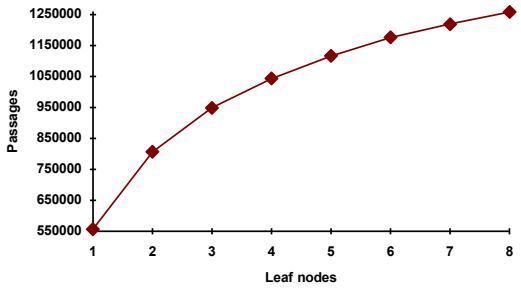


Fig 5. BASE1 [*title only*]: Retrieval efficiency, passages processed for 1000 documents per node

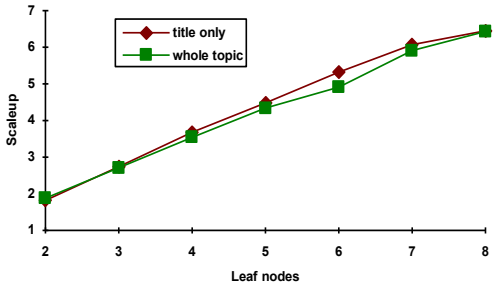


Fig 8. BASE1: Retrieval efficiency, scaleup for 1000 documents per node

The results on *title only* queries are very good. The elapsed time for every parallel

run on BASE1 is under a second (see fig 3) and meets the 10 second requirement suggested by Frakes [7]. The run using 8 leaf nodes is faster than 8 out of the 11 runs submitted by other participants at VLC2 [5]. Overall the results show a time reduction for BASE1 parallel runs over the sequential run, with a linear increase in throughput (see fig 4). Scaleup is super linear and increases with more leaf nodes in the topology (see fig 8): although the number of extra documents examined actually decreases with more leaf nodes (see fig 6). This increase in performance with respect to scale can be explained in part by the number of passages processed as the leaf node set is increased (see fig 5). Passages processed from 2 to 7 leaf nodes grow at a very slow rate, from just over half a million passages at 2 leaf nodes to 1.2 million inspected at 8. This effect occurs because more passage intensive documents are placed higher up the rank by the term weighting mechanism. The load imbalance figures demonstrate that the workload is fairly distributed amongst leaf nodes (see fig 7). It should be noted that the slight drop in relative performance on 8 leaf nodes for *title only* queries is because client processes had to be mapped to one of the search leaf nodes and this affected the timings very slightly.

The elapsed time for *whole topic* queries on BASE1 (see fig 9) do not meet the 10 second requirement for elapsed time suggested by Frakes [7] on runs up to 5 leaf nodes. Throughput is therefore much reduced compared to *title only* (see fig 10). The comparison with other VLC2 participants is not so good: the run at 8 leaf nodes only betters 2 out of the 11 submitted runs [5]. While run times are much slower than *title only*, the other evidence found in those experiments are confirmed in these. Scaleup is super linear (see fig 8) which is surprising since the number of documents processed is very near the maximum (see fig 6). As with *title only* the number of passages processed reduces with increasing the number of leaf nodes and for the same reason (see fig 11). The actual number of passages inspected is more with *whole topic* than with *title only* (see figs 5 and 11). Load imbalance is very small with *whole topic* queries (see fig 7).

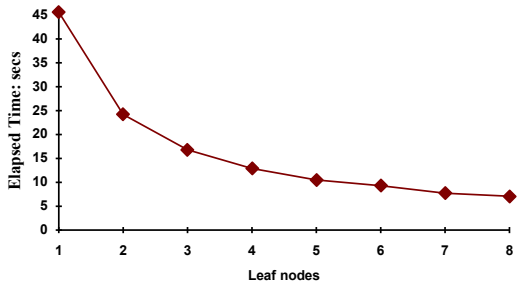


Fig 9. BASE1 [*whole topic*]: Retrieval efficiency, average elapsed time in seconds for 1000 documents per node

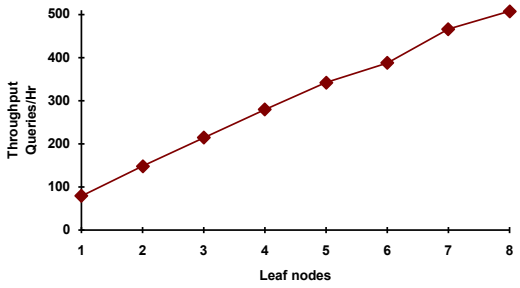


Fig 10. BASE1 [*whole topic*]: Retrieval efficiency, throughput for 1000 documents per node

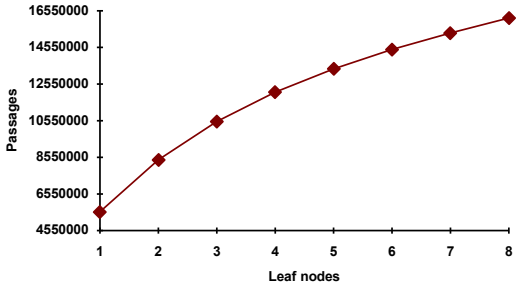


Fig 11. BASE1 [*whole topic*]: Retrieval efficiency, passages processed for 1000 documents per node

Concerning retrieval effectiveness on *title only* it is found that no increase accrued for precision at 5,10 and 15 and it decreases on precision at 20 as number of leaf nodes is increased (see table 2). There are also slight variations in *whole topic* results (see table 3).

However no decrease or increase found is statistically significant. We can state that examining the extra passages does not bring any benefit to any of the retrieval effectiveness measures used. It is also a clear indication that the ranking process is doing its job: the best documents useful for passage retrieval are contained in the top 1000 ranked documents.

Leaf nodes	**p@ 5**	**p@ 10**	**p@ 15**	**p@ 20**
1	0.268	0.186	0.161	0.148
2 to 4	0.268	0.186	0.161	0.147
5 to 8	0.268	0.186	0.161	0.146

Table 2. BASE1 [*title only*]: Retrieval effectiveness results for 1000 documents per node

Leaf nodes	**p@ 5**	**p@ 10**	**p@ 15**	**p@ 20**
1	0.196	0.160	0.140	0.126
2 to 8	0.200	0.162	0.141	0.127

Table 3. BASE1 [*whole topic*]: Retrieval effectiveness results for 1000 documents per node

Table 4 shows the comparison of BASE1 and BASE10 measures. Elapsed times in general are good apart from *whole topic* queries on BASE10: a minute or more response time for queries is not acceptable [7]. The average elapsed time for *title only* queries on BASE10 is better than half the runs of other VLC2 participants [5]. The scalability derived from BASE1 to BASE10 is particularly good for *whole topic* queries which recorded 1.08 (a figure of 1.0 is linear scalability). Throughput on BASE1 is much better than BASE10 as one would expect. All precision measures are better at BASE10 than BASE1: all record a reasonable increase. As with BASE1, there is little variation in BASE10 precision results for both types of queries.

MEASURE	**BASE1**		**BASE10**	
	title only	**whole topic**	**title only**	**whole topic**
Time (secs)	0.207	7.09	2.27	65.7
Throughput (Queries/Hr)	17,406	508	1,588	55
Scalability (1 to 10)	-	-	0.91	1.08
LI	1.052	1.005	1.015	1.003
p@ 5	0.268	0.200	0.320	0.348
p@ 10	0.186	0.162	0.304	0.310
p@ 15	0.161	0.141	0.275	0.275
p@ 20	0.146	0.127	0.251	0.249

Table 4. BASE1/BASE10: Retrieval effectiveness and efficiency results for 1000 documents per node (8 nodes).

In summary we state that while retrieval efficiency advantages are gained by using this type of parallelism (in spite of the extra passage data inspected), there is no clear gain in retrieval effectiveness. From the evidence given above the ranking process does its job well, therefore processing extra documents using our passage retrieval method is unnecessary when the BM25 weighting function is used. Hypothesis 2 is not supported given the evidence supplied by these experiments.

8.2 Experiment Results for 1000 documents overall

The results gained in these experiments for *title only* queries can only be described as remarkably good. The parallel measurements demonstrate this clearly with a super linear speedup recorded on all parallel runs (see fig 17). As a result all parallel efficiency results are greater than 1 with most figures greater than 1.5 (see fig 18). Load imbalance is very small with most figures very near 1 (see fig 16). All elapsed times are very good and under half a second (see fig 12). The scalability from BASE1 to BASE10 is very good, recording a figure of 1.32 (refer to table 5).

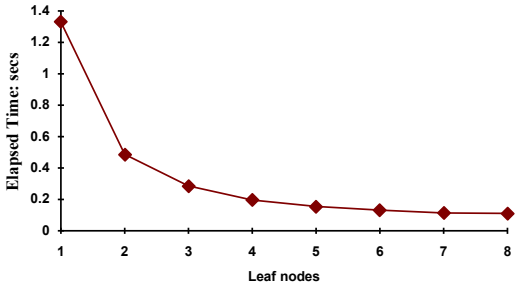


Fig 12. BASE1 [*title only*]: Retrieval efficiency, average elapsed time in seconds for 1000 documents overall

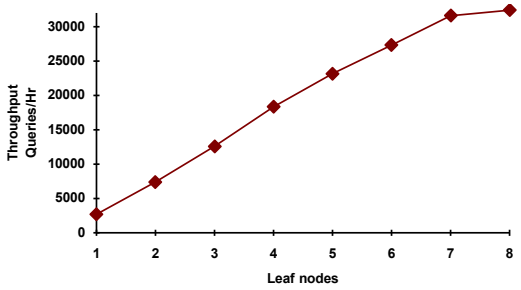


Fig 13. BASE1 [*title only*]: Retrieval efficiency, throughput (queries/ hour) for 1000 documents overall

The method gains substantially from very low communication overhead. Why does a substantial speed advantage over sequential processing occur with this method of parallel passage retrieval? There is no obvious effect from the number of passage processed (see fig 14), as the number processed fluctuates with leaf node count. This effect has to occur because the documents examined in paralleism are less computationally intensive to examine than the set examined on the uniprocessor i.e. each passage on the uniprocessor is on average more expensive to process than on the parallel nodes. There is an interesting interaction between the parallel algorithm and the weighting function that merits investigation.

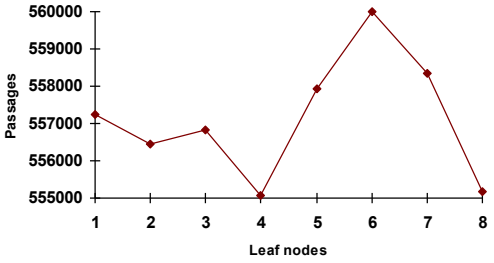


Fig 14. BASE1 [*title only*]: Retrieval efficiency, passages processed for 1000 documents overall

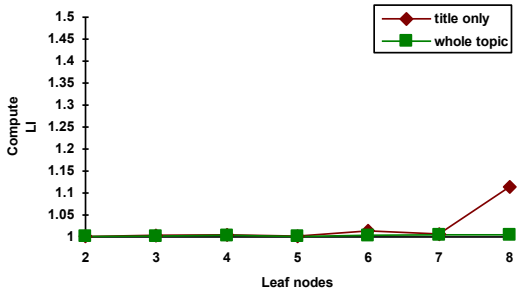


Fig 16. BASE1 [*title only*]: Retrieval efficiency, load imbalance for 1000 documents overall

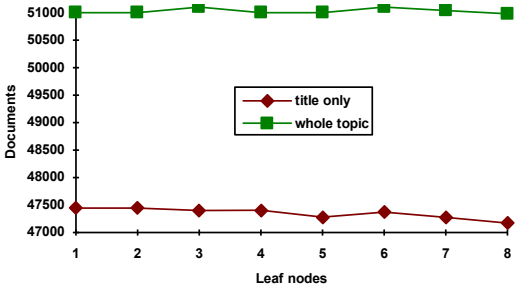


Fig 15. BASE1 [*title only*]: Retrieval efficiency, documents processed for 1000 documents overall

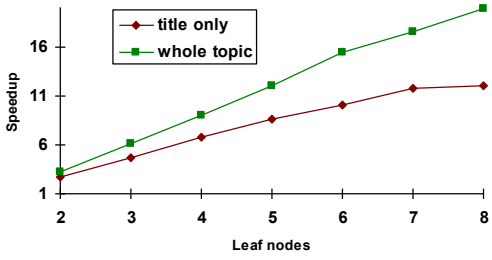


Fig 17. BASE1 [*title only*]: Retrieval efficiency, speedup for 1000 documents overall

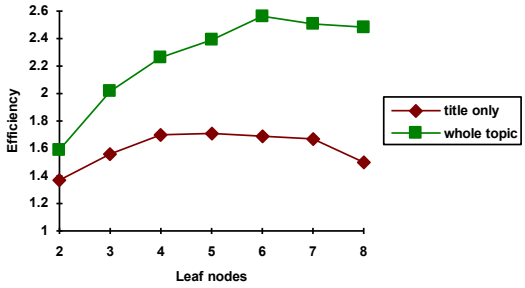


Fig 18. BASE1 [*title only*]: Retrieval efficiency, parallel efficiency for 1000 documents overall

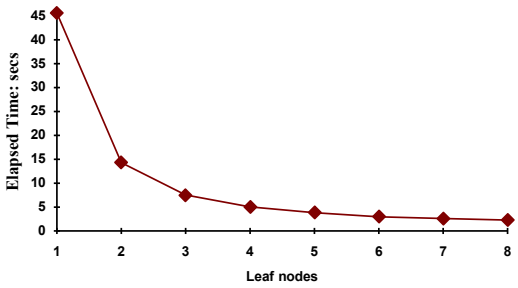


Fig 19. BASE1 [*whole topic*]: Retrieval efficiency, average elapsed time in seconds for 1000 documents overall

The query processing results are even more remarkable using *whole topic* queries than using the *title only* query set. A large part of the difference can be put down to the number of passages processed in these experiments as compared with a uniprocessor on *whole topic*: just over three million for the former compared with five and half million for the later (see fig 21). As with *title only* queries the numbers do not vary much on runs with differing numbers of leaf nodes. Response times for parallel runs are only unacceptable at 2 leaf nodes (see fig 19) and throughput is much improved in these experiments compared to uniprocessor experiments (see fig 20).

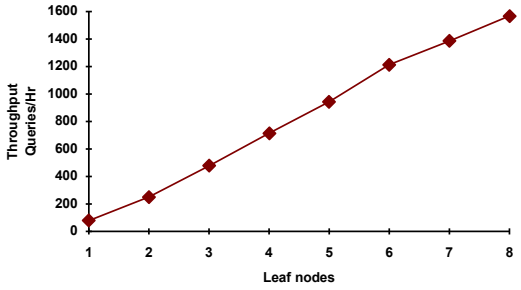


Fig 20. BASE1 [*whole topic*]: Retrieval efficiency, throughput (queries/ hour) for 1000 documents overall

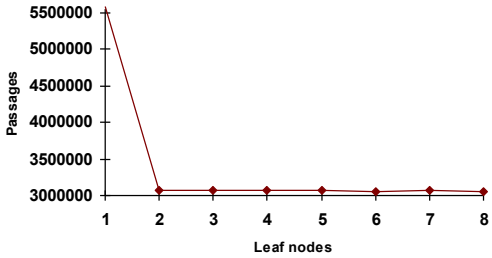


Fig 21. BASE1 [*whole topic*]: Retrieval efficiency, passages processed for 1000 documents overall

The retrieval effectiveness found on uniprocessor experiments is also found in these experiments for both types of queries, e.g. 0.148 for precision at 20 for *title only*. Precision at 20 varies very slightly for *whole topic* queries, but the difference is not significant, e.g. values of 0.126/0.127 were recorded. From this we deduce that examining the top 1000 documents on the uniprocessor is no different from examining 1000 documents by selecting top ranked documents locally on each leaf node. The evidence from this leads directly on to experimentation described in the next section.

8.3 Experiment results for reduced document sets

As we have observed from the above, it does not seem to matter if we examine 1000 documents or more, retrieval effectiveness at lower precision points is not improved or harmed much by any of the methods applied. We decided to experiment with a smaller set of documents using the parallel passage retrieval algorithm on 8 leaf nodes with the BASE1 and BASE10 collections. This is done to see how the reduction affected the performance of the parallel algorithm. We chose 504, 256 and 128 being near a half, a quarter and eighth respectively of experiments on 1000 documents overall (we include no passages results in tables 5 to 8 for comparison purposes).

Docs pro-cessed	p@ 5	p@ 10	p@ 15	p@ 20
1000	0.268	0.186	0.161	0.148
504	0.268	0.186	0.161	0.147
256	0.268	0.184	0.160	0.146
128	0.268	0.184	0.159	0.149
0	0.244	0.178	0.149	0.130

Table 5. BASE1 [*title only*]: Retrieval effectiveness results on varying the number of documents in which passage processing is applied.

Docs pro-cessed	p@ 5	p@ 10	p@ 15	p@ 20
1000	0.200	0.162	0.141	0.127
504	0.200	0.162	0.141	0.127
256	0.200	0.162	0.141	0.127
128	0.196	0.160	0.140	0.126
0	0.188	0.172	0.145	0.128

Table 6. BASE1 [*whole topic*]: Retrieval effectiveness results on varying the number of documents in which passage processing is applied.

Docs pro-cessed	p@ 5	p@ 10	p@ 15	p@ 20
1000	0.324	0.302	0.275	0.254
504	0.324	0.302	0.275	0.254
256	0.324	0.304	0.276	0.252
128	0.320	0.300	0.279	0.253
0	0.324	0.282	0.273	0.264

Table 7. BASE10 [*title only*]: Retrieval effectiveness results on varying the number of documents in which passage processing is applied.

The evidence given in tables 5 to 8 demonstrates that we do not need to examine the full 1000 documents to achieve very nearly the same level of retrieval effectiveness at lower precision. There are some differences but they are very minor for both types of query. We can reduce the level of computation on the passage retrieval method and still obtain the extra retrieval effectiveness found when using that method. The experiments reinforce the assertion that the BM25 ranking function does its job well. This function in conjunction with passage retrieval applied to smaller document sets can improve retrieval effectiveness.

From table 9 we can see that elapsed times for *title only* queries change only very slightly as the number of documents examined by the passage retrieval decreases. The times in seconds from term weighting searches without passage processing (0.06 for BASE1, 0.54 for BASE10), show that while searches on BASE10 show linear reduction from 1000 to 128, those on BASE1 do not. This indicates that the size of collection combined with the term weighting scheme are significant factors that affect passage retrieval performance. As BASE10 is a much larger database, searches on it will pick documents that are roughly the same in passage processing costs and they are ranked in the top set by the term weighting scheme. Searches on BASE1 generate more passage intensive documents higher up the rank, which require extra computation: further evidence that the ranking process is doing its job. Other measures such as scalability and LI are good.

Docs pro-cesse d	p@ 5	p@ 10	p@ 15	p@ 20
1000	0.352	0.310	0.275	0.251
504	0.356	0.312	0.271	0.249
256	0.356	0.304	0.272	0.246
128	0.364	0.306	0.267	0.246
0	0.356	0.298	0.271	0.247

Table 8. BASE10 [*whole topic*]: Retrieval effectiveness results on varying the number of documents in which passage processing is applied.

Measure	docs=1000		docs=504		docs=256		docs=128	
	BS1	BS10	BS1	BS10	BS1	BS10	BS1	BS10
Time (secs)	0.11	0.84	0.10	0.71	0.09	0.62	0.08	0.58
Throughput (Queries/Hr)	32k	4.2k	38k	5.1k	42k	5.8k	46k	6.2k
Scalability	-	1.32	-	1.35	-	1.37	-	1.34
LI	1.11	1.04	1.14	1.04	1.16	1.04	1.18	1.05

Table 9. BASE1/BASE10 [*title only*]: Retrieval efficiency results for on varying numbers of documents

Measure	docs=1000		docs=504		docs=256		docs=128	
	BS1	BS10	BS1	BS10	BS1	BS10	BS1	BS10
Time (secs)	2.29	16.3	1.64	10.4	1.28	8.47	1.09	7.50
Throughput (Queries/Hr)	1.5k	221	2.2k	346	2.8k	425	3.3k	480
Scalability	-	1.41	-	1.58	-	1.52	-	1.45
LI	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01

Table 10. BASE1/BASE10 [*whole topic*]: Retrieval efficiency results for on varying numbers of documents

The figures for *whole topic* queries show more improvement than *title only* queries with respect to average query processing time (see table 10). The timings from term weighting searches without passage processing (0.77 for BASE1, 6.45 for BASE10) show that the time reduction is very near linear. Load balance for all runs is very good. The scalability is excellent for all BASE10 runs, and confirms the effect of collection size and term weighting function on performance.

9. Summary and conclusion

The retrieval efficiency results given in this paper is in the main very good indeed: the performance improvement using parallelism is very good on both types of query. Complexity analysis suggest that the passage retrieval algorithm used in this research is reduced from O((a(a-1)/2)) on a uniprocessor to O((a(a-1)/2)/P) when parallelism is used (where a is the number of text atoms and P is the number of leaf nodes). All parallel measures show an increase in performance that infers this complexity analysis is justified. The elapsed time for any run on *title only* queries were well under the 10 seconds recommended by Frakes [7], while the use of parallelism on *whole topic* queries does yield acceptable run times when using a larger node set. To the best of our knowledge we are the first to make a practical attempt at speeding up the passage retrieval algorithm described in this paper. We have demonstrated that hypothesis 1 on retrieval efficiency has been confirmed, that is we have demonstrated speedup and scaleup for passage retrieval using

parallel computing.

With respect to retrieval effectiveness, the passage retrieval algorithms do bring benefits on web data over ordinary term weighting but such is not guaranteed. However it is clear from the above that the algorithm only needs to be applied to a subset of the top ranked documents to gain effectiveness. Clearly the ranking process using BM25 does its job very well. However gains are collection and query set dependent (this result is confirmed in our TREC8 experiments [8] and independently confirmed by Kazkiel & Zobel [9]). With *title only* queries on the BASE1 database yield an increase of 13.8% from 0.130 to 0.148 in precision at 20 when passage retrieval is applied. A reduction in effectiveness on some precision points is recorded when the same queries are applied to the BASE10 collection: any increase found is not significant. *Whole topic* queries do not do as well as *title only*, a problem that merits further investigation.

While retrieval efficiency improvement is significant using parallelism on passage retrieval clearly many of the experiments described above to do not bring any benefits with respect to effectiveness over the sequential passage retrieval algorithm. A hypothesis which asserts that examining more of the search space in passage retrieval yields more relevant documents is not supported by the results given in this paper - indeed, the evidence is against it. This is not to suggest that applying parallelism to passage retrieval does not work, but the choice will depend on whether to use parallelism with term weighting or not. We can apply the passage retrieval algorithm to fewer documents, at a computational cost slightly higher than that term weighting to obtain better retrieval effectiveness. The implication is that parallelism is not worth applying to passage retrieval on a reduced set of documents, if it is not worth applying the strategy to term weighting.

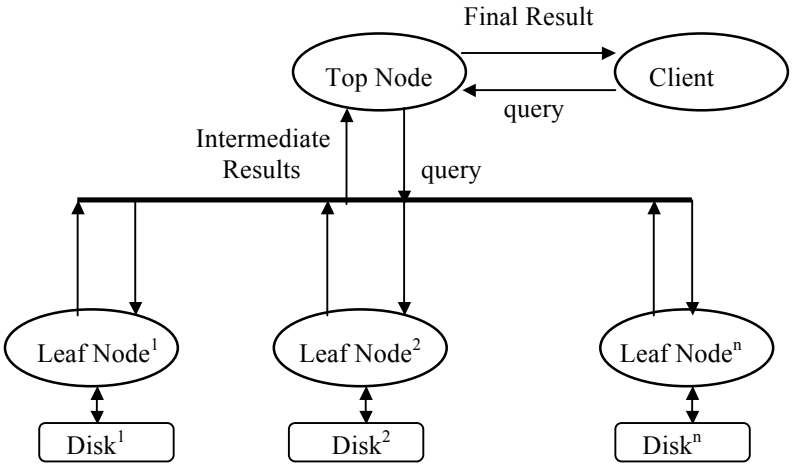
10. Acknowledgements

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

References

1. A. MacFarlane, Inverted files and performance: A study of parallelism and data distribution methods in IR, PhD thesis, City University London, 2000.
2. S.E. Robertson, S.Walker, S. Jones, M.M. Hancock-Beaulieu and M Gatford, Okapi at TREC-3. In: D.K.Harman, (ed.): *Proceedings of the Third Text Retrieval Conference*, NIST Gaithersburg (1995) 109-126.
3. A. MacFarlane, S.E. Robertson, and J.A. McCann. *Parallel Methods for the Search of Partitioned Inverted Files*, In: P. De La Fuente, (ed.), SPIRE 2000: Proceedings of String Processing and Information Retrieval, September 2000, A Coruna, Spain, (IEEE Computer Society Press, Los Alamitos, 2000).
4. 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*, 31(3) (1995) 345-360.
5. D., Hawking, N. Craswell, and P. Thistlewaite. Overview of TREC-7 Very Large Collection Track. In: Voorhees E., and Harman, D.K., (eds.), *Proceedings of the Seventh Text Retrieval Conference, Gaithersburg, U.S.A, November 1998*, NIST SP 500-242 (NIST, Gaithersburg, 1999).
6. S.E. Robertson and K. Sparck Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science* May-June (1976) 129-145.
7. W.B. Frakes Introduction to Information Storage and Retrieval Systems. In: W.B. Frakes, and R. Baeza-Yates (eds.), *Information Retrieval: Data Structures and Algorithms*, (Prentice-Hall, New Jersey, 1992).
8. A. MacFarlane, S.E. Robertson & J.A. McCann. PLIERS at TREC8. In: E. Voorhees and D.K. Harman (eds.), *Proceedings of the Eighth Text Retrieval Conference, Gaithersburg, U.S.A, November 1999, Gaithersburg*, NIST SP 500-246, (NIST, Gaithersburg, 2000).
9. M, Kaszkiel, & J. Zobel. Effective ranking with arbitrary passages. *Journal of the*

Appendix 1 - Example of search topology used



Appendix 2 – The BM25 Term Weighting Function

$$CW(i,j) = \frac{CFW(i) * TF(i,j) * K1+1}{K1 * ((1-B)+(B*(NDL(j)))) + TF(i,j)}$$

Variables

CW(i,j) : Weight for term t(i) in document d(j).
CFW(i,j) : Collection frequency weight log(N) - log(n).
n(i) : The number of documents term t(i) occurs in.
N : The number of documents in the collection.
TF(i,j) : The number of occurrences of term t(i) in document d(j). (term frequency)
DL(j) : The total number of terms in document d(j)
NDL(j) : Normalised document length (DL(j) / average DL for all documents.)

Constants

K1 : Constant that modifies influence of term frequency.
B : Constant that modifies effect of document length.