Parallel Computing for Term Selection in Routing/Filtering

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 CB2 3NH
[3]Department of Computing, Imperial College London

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

Abstract: It has been postulated that a method of selecting terms in either routing or filtering using relevance feedback would be to evaluate every possible combination of terms in a training set and determine which combination yields the best retrieval results. Whilst this is not a realistic proposition because of the enormous size of the search space, some heuristics have been developed on the Okapi system to tackle the problem which are computationally intensive. This paper describes parallel computing techniques that have been applied to these heuristics to reduce the time it takes to select to select terms.

1. Introduction

This paper describes parallel computing techniques that can be applied to a very large search space for relevance feedback for routing/filtering that has successfully been used on Okapi experiments at TREC [1-4]. The routing task we are considering is the situation where a number of relevance judgements have been accumulated and we want to derive the best possible search formulation for future documents. The filtering task is an extension of this, but a threshold for documents is applied i.e. a binary yes/no decision is made on one document at a time as to whether it will be presented to the user. In [1] it was stated that an alternative to some term ranking methods described would be to "evaluate every possible combination of terms on a training set and use some performance evaluation measure to determine which combination is best". Such a method is very computationally intensive, and certainly not practicable even with parallel machinery. The complexity of one routing session before re-weighting is 2^t where t is the number of terms. For just 300 terms, our search space is 2^{300} or 2.04e+90 combinations. Since the term scores also can be re-weighted any number of times, the order for the algorithm's time complexity cannot be stated. Clearly some sort of limit must be set both on the number of times re-weighting is allowed and the total number of combinations inspected for an information need. We use combinatorial optimisation techniques on this search space and apply parallelism to improve the speed of the techniques studied. The methods used by Okapi at TREC are briefly described in section 2 and the strategy for applying parallelism to these methods is described in section 3. The data and settings used for the experiments are described in section 4. Section 5 describes the experimental results. Conclusions and further work in the area are outlined in section 6.

2. Methods used by Okapi at TREC

Okapi at TREC [1-4] applied three algorithms to the term selection problem. Find Best (FB), Choose First Positive (CFP) and Choose All Positive (CAP). With the FB algorithm each term is evaluated in one iteration and the term yielding the best score from the term set chosen: the algorithm stops when a pre-determined threshold is reached. An evaluation in this context is a retrieval followed by the application of some function on the search results to produce a score: for example average precision. Terms that increase this score are retained/removed (see below). The threshold that halts the procedure is reached when there is little or no increase in the score. The other algorithms work in much the same way, but with minor differences. CFP works by choosing the term if it is the first term that increases the score. CAP is an extension of FB/CFP and works by including/excluding all terms that increase the score in an iteration. Each chosen term is accumulated in a *query set*, the final version of which is applied to a test set. Within each algorithm, two operations for choosing terms can be used: add term to the query or delete term from the query. The add term operation can be augmented by reweighing the retrieval selection value: in the case of the Okapi experiments this is either a reduction by a factor of 0.67 or an increase by a factor of 1.5. The Find Best and Choose All Positive algorithms are Steepest-ascent Hill-Climbers while Choose First Positive is a First-ascent Hill-Climber [5].

3. Applying parallelism to the Okapi methods

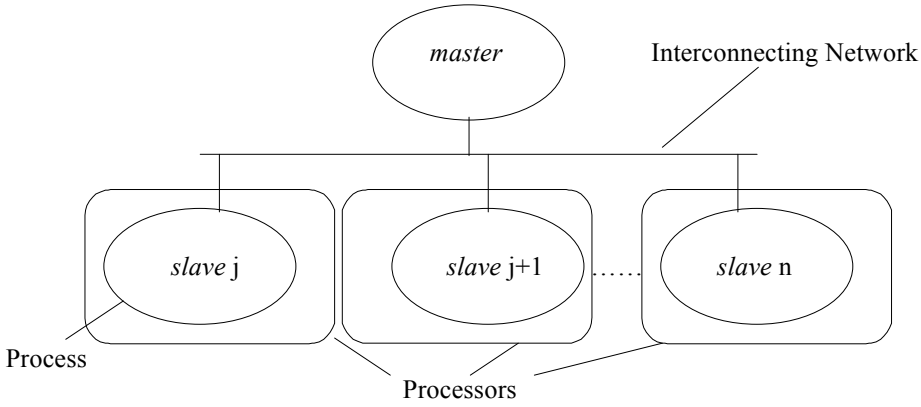

Fig 1 - *Master*/*Slave* Router Topology

An approach to applying parallel computation to term selection is to think in terms of *term set*s and what needs to be done to the Hill-Climber algorithms to reduce their run time. Since the evaluation of operations on terms can be done independently we can distribute the *term set* to a number of slave processes which apply the required Hill-Climber algorithm to each sub-set of the *term set*: an individual term is only evaluated on one processor. Thus by applying inter-set parallelism to the evaluation of terms in the evaluation set, we aim to speed up each iteration. We use a method of parallelism known

as the *domain decomposition strategy* [6]: the search space is divided amongst available slave processors, controlled by a master process (see fig 1). One of the advantages of this method is that communication costs are kept to a minimum as processes involved in evaluating terms do not need to communicate to complete their task: however there is an overhead associated with checking stopping criterion in every iteration. This overhead involves both the retrieval of the best yielding term or terms from all slaves by the master and broadcast of the best term data back to the slaves. Each slave has access to the training set on its own local disk in order to increase the flexibility of the parallel algorithms and reduce the overheads of broadcasting term information from the master process to slaves. This method of data distribution is known as *Replication*.

The combinations of algorithms and operations described in this paper are: Find Best, Choose First Positive and Choose All Positive algorithms with *add only*, *add/remove* operations and *add with re-weighting*. It should be noted that the CAP algorithm is a purely sequential algorithm to which inter-set parallelism cannot be directly applied, as the results are the cumulative effect of evaluations in one iteration. However the CAP algorithm can be applied to each sub-set of the term set and we refer to revised version as the Choose Some Positive (CSP) algorithm: we can regard CSP as a compromise between the FB/CFP algorithms and CAP algorithm. In the CSP algorithm the best yielding sub-set of the term set from one process only is chosen. CSP is implemented in terms of CAP. Choose First Positive differs slightly in that it is possible that a better term could be chosen in one inner iteration for each smaller sub-set of the term set (or increasing number of processes). It is possible the terms selected by the Find Best algorithm may differ slightly over runs with varying numbers of processes, possibly affecting the evaluation score. This is because two or more terms may have the same effect when applied to the query and the term that is chosen first amongst these equal terms will be the term used. When the number of processors equals the number of evaluation terms, all term selection algorithms are identical; i.e. they all reduce to Find Best.

4. Description of the data and settings used in experiments

The database used for the experiments was the Ziff-Davis collection from the TREC-4 disk 3 that is 349 Mb in size with a total number of 161,021 records [7]. Three databases were created for the Ziff-Davis collection: an extraction database, a selection database (both of which form the training set) and a test database. Our focus here is on the training set. We use the extraction database to choose the initial set of terms for the queries using the relevance judgements, and then train the queries on the selection database using the Okapi Hill-Climbers. We used a total of 19 TREC-4 topics on the Ziff-Davis database for these experiments. These topics were chosen on the basis of the number of relevance judgements available for routing/filtering: it was felt that topics with too few relevance judgements (i.e. one or two) would not be of much use in the optimisation process due to excessive overfitting (overfitting can occur when term selection mechanism overtrains). The distribution of relevance judgements for the database was as follows; 1868 (39%) for the extraction set, 1469 (30%) for the selection set and 1483 (31%) for the test set.

The timing metrics we use to measure retrieval efficiency are as follows. For each run we declare the average elapsed time in seconds for term selection over all topics.

We define selection efficiency as the improvement in average elapsed time by using parallelism. We use the standard parallel measures:

- Speedup: defined as the increase in speed from 1 to n processors and found by dividing time spent on computation using 1 processor by time using n processors.
- Parallel efficiency: defined as speedup divided by n processors giving an idea of how well processors are being used on a parallel system.
- Load imbalance: we use a metric called LI that is the ratio of the maximum elapsed time over all the processors divided by the average elapsed time [8]: a perfect load balance would achieve an LI of 1.0.

As our focus is on speeding up the algorithms, we do not discuss retrieval effectiveness issues: experiments done by Okapi at TREC have shown that an increase in retrieval effectiveness is available [1-4]. We present runs on 1 to 7 processors: the hardware used for the research was the Fujitsu AP3000 at the Australian National University.

5 Experimental results

5.1 Elapsed time for term selection

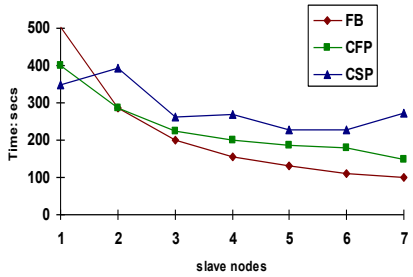

Fig 2. *Add only* average term selection elapsed time in seconds

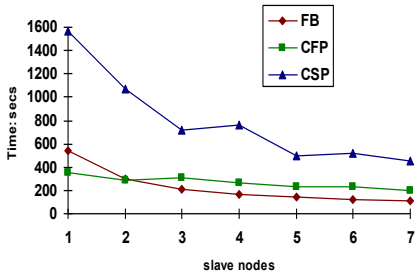

Fig 3. *Add remove* average term selection elapsed time in seconds

From figs 2 to 4 it can be seen how expensive the application of the term selection algorithms can be, with average elapsed time running into hundreds of seconds and in some cases thousands of seconds. Parallelism has different effects on individual term selection methods which can be either beneficial or detrimental. The FB algorithm is the term selection method which benefits most from the application of parallelism, showing a linear time reduction on all node set sizes. FB also outperforms the other term selection algorithms using more processors (this can be seen from all data presented in figs 2 to 4). Linear time reductions are also found with most parallel runs on CFP using any operation (this is most noticeable with *add only* operation - see fig 2). With regard to CSP using any operation, elapsed times do not follow any trend and vary unpredictably with slave node set size (particularly using *add only* operation - see fig 2). The most expensive operation in the majority of cases is *add reweight*: for example FB run times are roughly four times as slow on *add reweight* as the other operations. It is generally

more expensive to use the *add/remove* operation compared with *add only* particularly with the FB algorithm.

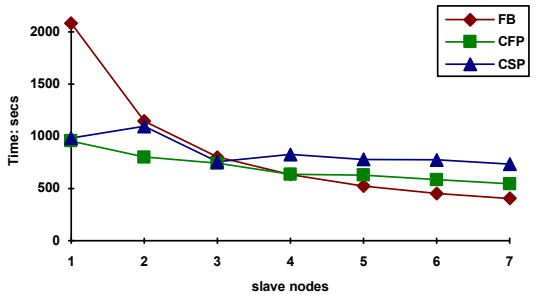

Fig 4. *Add reweight* average term selection elapsed time in seconds

5.2 Load imbalance

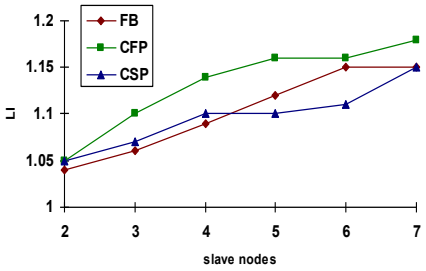

Fig 5. *Add only* load imbalance for term selection

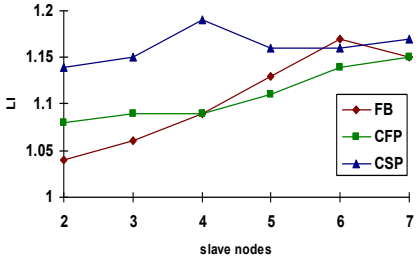

Fig 6. *Add remove* load imbalance for term selection

The imbalance for term selection is low and does not reach a point where load balance is a significant problem for the algorithms: for example an LI of 2.0 would mean halving the effective speed of the machine and the LI figures in figs 5 to 7 are nowhere near that level. However, general trend for load imbalance for most experiments is upwards. The exception is CSP with *add remove* operation which shows a reduced level of load balance over all runs. There is a clear increase in load imbalance as the number of slave nodes is increased, which demonstrates the need for some form of load balancing technique if many more slave nodes were to be used in optimising on a training set of this size. This imbalance contributes in part to the overall loss in term selection efficiency recorded.

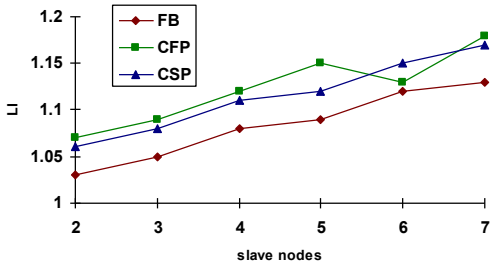

Fig 7. *Add reweight* load imbalance for term selection

5.3 Speedup and parallel efficiency

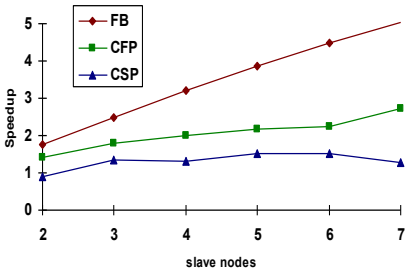

Fig 8. *Add only* operation speedup for term selection

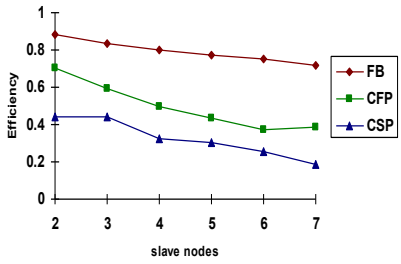

Fig 9. *Add only* parallel efficiency for term selection

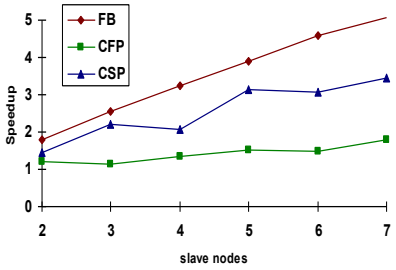

Fig 10. *Add remove* operation speedup for term selection

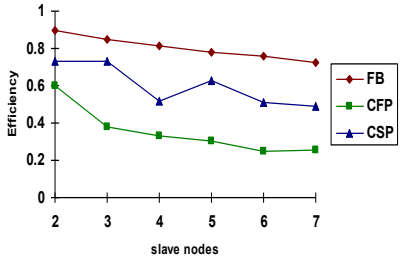

Fig 11. *Add remove* operation parallel efficiency for term selection

The speedup and efficiency figures are shown in figs 8 to 13. In terms of speedup and parallel efficiency, the FB method shows improvement on all levels of parallelism investigated. Speedup is near linear at 7 slave nodes with parallel efficiency above the 70% mark for any operation. However the speedup and parallel efficiency for CFP is very poor for all three term operations. In most cases a speedup of less than two is registered: a number of factors are responsible for the poor parallel performance. An increase in evaluations with more slave nodes is a significant factor as well as the overhead at the synchronisation point together with load imbalance. For example CFP

with *add reweight* increases the evaluations per topic from 3787 on 1 slave node to 5434 on 7 slave nodes: the same trend is found with other operations.

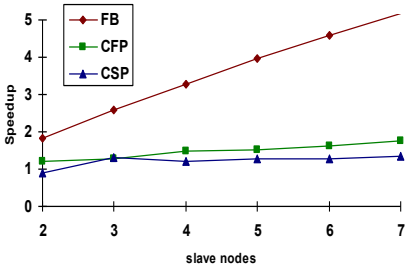

Fig 12. *Add reweight* operation speedup for term selection

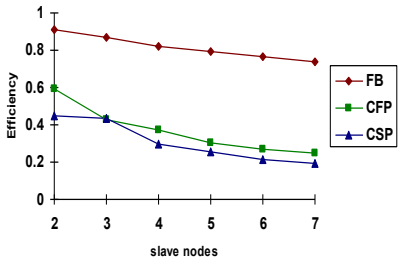

Fig 13. *Add reweight* operation parallel efficiency for term selection

Much the same can be said for CSP, apart from *add remove* operation which does actually show some level of speedup. However, overheads are a much less significant factor for CSP while the increase in evaluations plays a more important part: for example the number of evaluations using the *add reweight* operation increased from 2766 per topic on 1 slave node to 8234 on 7 slave nodes. There are fewer iterations with the CSP method, but individual iterations are much longer. Slowdown for CSP on *add only* and *add reweight* is recorded for 2 slave nodes. It could be argued that using speedup and parallel efficiency to measure the parallel performance of the CSP algorithm is unfair as the parallelism itself imposes an extra workload for the method. However demonstrating that some parallel performance improvement is available while still being able to examine some of the search space is, we believe, worthwhile.

6. Conclusion and further work

We have found a method of parallelism which by focusing on the main task, namely the evaluation of terms, can speed up the computation of the heuristics and examine more of the search space. We have shown that the speed advantage found with the FB selection method is significant. We believe it is possible to improve the selection efficiency of both FB and CSP using some form of dynamic re-distribution technique for terms in the query. Experiments with CFP are less conclusive and show difficulties particularly with load balance. It may be possible to improve the load balance of CFP but only at a large overhead cost.

We could consider the use of *machine learning* [9] *tabu* search [6] and *pattern recognition* [10] techniques in order to optimise routing/filtering queries. A great deal of research into search space methods has been done in *machine learning* using methods such as genetic algorithms and neural networks that are both very computationally intensive processes. *Tabu* search is a meta-heuristic which can be used to manage other heuristics in order to examine parts of the search space which would not normally be examined with a single search strategy. Some of the selection algorithms used in *pattern recognition* are similar to the Hill-Climbers used in this study [10], particularly Find Best with *add only* and *remove only* operations. We could therefore treat the query optimisation discussed in this research as a pattern recognition problem, treating different

combinations of the query as a pattern. The problem would be to find the best yielding 'pattern' in the query. Parallelism could be used to speed up these methods, providing they are able to show retrieval effectiveness benefit on the test set.

7. Acknowledgements

This work was supported by the British Academy under grant number IS96/4203. We are grateful to the Australian National University, Canberra for the of their Fujistsu AP3000 parallel computer in order to conduct these experiments. We owe particular thanks to Gordon Smith, David Hawking and David Sitsky for their advice on many issues. We would also like to thank David Hawking for suggesting the use of replication for the method of data distribution.

References

1. 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
2. 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*, , NIST Gaithersburg (1996) 73-96
3. M.M. Beaulieu, M. Gatford, X. Huang, S.E. Robertson, S. Walker and P. Williams, Okapi at TREC-5. In: Voorhees E.M and D.K.Harman, (eds.): *Proceedings of the Fifth Text Retrieval Conference*, NIST Gaithersburg (1997) 143-166.
4. S. Walker, S.E.Robertson and M Boughanem, OKAPI at TREC-6. In: Voorhees E.M and D.K.Harman, (eds.): *Proceedings of the Sixth Text Retrieval Conference*, NIST Gaithersburg (1998) 125-136
5. A. Tuson, Optimisation with Hillclimbing on Steriods: An Overview of Neightbourhood Search Techniques. *Proceedings of the 10th Young Operational research Conference*, Operational Research Society (1998) 141-156
6. F. Glover and M. Laguna, Tabu Search, Kluwer Academic Publishers, 1997.
7. D. Harman, Overview of the Fourth Text REtrieval Conference (TREC-4). In: D.K.Harman, (ed.): *Proceedings of the Fourth Text Retrieval Conference*, NIST Gaithersburg (1996) 1-24
8. D. Hawking, "The Design and Implementation of a Parallel Document Retrieval Engine", Technical Report TR-CS-95-08, Department of Computer Science, Australian National University (1995)
9. A. Hutchinson, Algorithm Learning, Clarendon Press (1994)
10. J. Kittler, Feature selection and extraction. In: T.Y. Young and K. Fu, (ed.): *Handbook of Pattern Recognition and Image Processing*, Academic Press (1986) 59-83