Parallel Computing for Information Retrieval and Performance Issues

Andrew MacFarlane

1. Introduction

The use of performance models when considering the deployment of parallel computing to information retrieval applications is a much neglected area. During the late 1980's and early 1990's there was a great deal of interest in research into parallel computing for Information Retrieval, but due to various issues interest has faded and very little research is now being done MacFarlane (2000). Much of this work was a) empirically based and did not try to produce models of performance in order to obtain some kind of theoretical underpinning for the research; b) only tackled one issue i.e. search for the most part; and c) only looked at one method for the distribution of data to nodes in a parallel machine. The PhD thesis written by the author MacFarlane (2000) attempted to tackle these issues and succeeded with issue c), but a great deal still needs to be addressed in a) and b). In this chapter we will briefly review the relevant literature, present a model of performance developed in MacFarlane (2000) and outline areas of further research that the author regards as being fruitful. The chapter is organised as follows. Section 2 justifies the use of parallel computing to solve information retrieval problems. Section 3 reviews previous work in modelling parallel IR. In section 4 a definition of tasks in IR under discussion and describes a number of distribution methods for inverted files which could be used in these tasks. Using these defined tasks and distribution methods, a synthetic model of performance outlined in section 5, which is in turn examined using empirical evidence in section 6. Finally we draw conclusions from the material and outline further research.

2. Why parallel IR?

There is a limit to the gains which can be achieved algorithmically and while great strides have been made in the power of computer hardware parts (particularly CPU's) there will always be an absolute limit. This combined with increasing amounts of information being made available, particularly on the Internet, is putting increasing strain on sequential processing systems. The only way forward for many compute heavy tasks is to use parallelism. There are a number of computationally heavy tasks in information retrieval which do require the use of parallelism (more detail of these tasks is given in section 5). There is increasing interest in using much larger collections for experimentation recently with the introduction of a Terabyte track within the TREC series of conferences (Clarke et al, 2005) and some kind of parallelism will be useful to the participants.

It should be noted that the use of parallelism for IR caused some controversy in the late 1980's with criticism from Stone (1987) and Salton & Buckley (1988). The basis of this criticism was some work done on the massively parallel Connection Machine which used a signature file method (a fixed length surrogate is used to represent each document). Of the criticisms Stone's was more useful in that he put forward an alternative parallel method based on inverted files – a far more efficient method for search (Harman et al, 1992). Stones ideas have proved to be very influential and most parallel IR systems use inverted files. In a review, Rasmussen (1992) put forward a number of reasons for applying parallelism to IR problems:

- *Response times*: there may be situations (the web is a good example of this) where many users require access to the same document collection. The purpose of parallelism would be to reduce contention between queries thereby increasing the system response time and query throughput.
- *Very large databases*: the time to process queries increases linearly with the document collection e.g. a query on a 10 gigabyte collection which takes 10 times

longer than a query on a 1 gigabyte collection. The purpose of parallelism is to scale up the methods used to handle much larger collections.

- *Superior algorithms*: there are some models such as the extended Boolean models (Fox et al, 1992) which offer improved retrieval effectiveness at the cost of extra computation. The role of parallel computing is to make these algorithms useable in a realistic search environment.
- *Search cost:* Stanfill et al (1989) showed the resources needed to search a database approach a level of cost effectiveness, given the assumption that search time is linear with collection size and resource costs are static. The role of parallelism is to make the deployment of hardware economically effective (web search engines use this factor to very good effect).

In this chapter we concentrate on addressing the issue of *Response times*. The examples we give above are all related to queries, but the issue applies as much to other tasks in IR such as Indexing and Inverted file maintenance.

3. Review of previous work

There have been two major reviews of parallel computing for information retrieval, namely Rasmussen (1992) and MacFarlane et al (1997). For the most part these reviews concentrated on the practical implementation of information retrieval systems, such as the architectures and algorithms used, and models implemented etc. The Rasmussen (1992) review did have a section on evaluation performance using such metrics as Speedup, but does not tackle the issue of modelling performance. Performance modelling in this context means a predictive model that can be used to estimate how a given algorithm will behave under a given set of conditions. Of particular concern is the lack of recent interest in doing research in parallel IR: it is seen as a 'solved problem' etc. However there has been success in applying parallelism to real world IR problems: the most significant example is that of the Internet search engines. There is therefore good reason to produce predictive models of performance.

A few attempts haven been made to produce such models, but they also have limitations. Those models that provide a general performance overview of IR do not deal with the problem of distribution (Cardenas, 1975; Fedorowicz, 1987; Wolfram, 1992a and 1992b). Distribution in this context is the method of allocation of index data to nodes in a parallel computer. Much of the work described in the literature on the subject which does look at distribution either tackles one task (Jeong and Omiecinski, 1995; Tomasic and Garcia-Molina, 1993a & 1993b) or one aspect of a task such as the consideration of only one distribution method (Ribeirio-Neto et al, 1999; Hawking, 1996). We define a task as being a specific aspect of an IR system that has its own functionality (see section 5 below). In an attempt to address this issue MacFarlane (2000) produced a synthetic model of performance which tackles both the issue of distribution and on a number of different tasks. It is this model which is discussed in this chapter. It should be noted that we draw a distinction between synthetic models that only predict the relative difference between two algorithms and analytical models which actually try to predict real performance of the two algorithms. Attempts by the author to produce an analytical model of performance failed due to the complexity of trying to model real performance on a disparate number of tasks: this is the main reason that the issue has not be tackled satisfactorily in the literature.

In order to set the scene for our attempt at solving this problem we describe the issue of distributing inverted file data and define some tasks that we attempt to model using our techniques.

4. Distribution methods for inverted file data

We look at four distribution methods; *On-the-fly* distribution, *Replication* and two types of *Partitioning*. Because of the criteria used some data distribution methods are invalid for some tasks (this issue is tackled when the tasks are defined below). In order to explain the differences between these distributions we use an abstract task, which is defined as follows.

We have some inverted file data D, together with some work W to be done on D (W can be any information retrieval task e.g. index a document or do a search). When W is applied to D, we get a result R. Any or all of the three variables D, W and R may be subdivided or partitioned in some way; W', D' and R' will denote some such partition or subset, which may nevertheless be the whole of the original variable in some cases. We define some algorithmic steps on these variables that are common to all distribution methods;

- A central node sends W" plus any data needed to a number of i identical processor nodes.
- The processor nodes apply W' to D' to produce results R'.
- The results R' are sent back to the central node which prepares the final result R using all R' results.

It may help the reader to think of an inverted file (or more formally D) as a matrix with the rows made up of term references and the columns made up of references to documents, see fig 1.

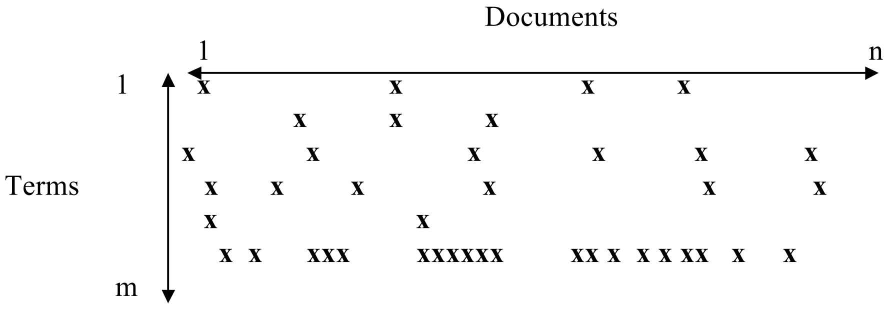

Fig 1: Example matrix

In the example we have m terms in the collection, which invert n documents from the indexed text. Relations between terms and documents are signified by a **x**. The matrix will be very sparse, and some relations between documents and terms or vice versa will be richer than others. We will use this matrix representation to show how data is distributed in the methods to be examined in this chapter.

4.1 *On-the-fly distribution*
We define *On-the-fly* distribution as the distribution of part of the data as it is required by a given task. It is a dynamic data distribution method, all the others are static. The method is very flexible in that we can distribute the whole matrix D or any D' which could be the whole or part of either a row or column. The inverted file data is held in one location. Consider the following example;

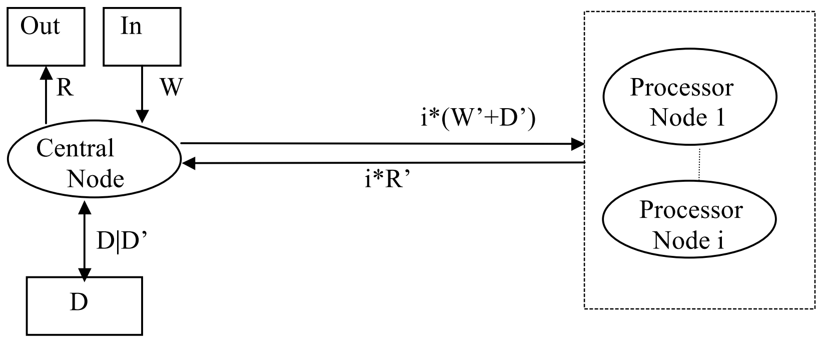

Fig 2: Example of abstract task on On-the-fly distribution

Note that the ellipse is a node, a box is a disk, and the dashed line box is the universe of our processor nodes. The arrows signify the direction of data exchange between nodes and or disks. Our abstract task uses the following algorithmic steps. The work W is input and the data associated with W (D') is lifted from the inverted file and is packed up with a subset of W (W'). Each of the i processor nodes each gets its own W' and D', and processes the data producing R'. All results from i processor nodes (R') are sent back to the central node which prepares the final result R. The significant disadvantage with this method is that a large amount of data must be distributed before computation can be done. The communication may swamp any useful work, and this makes the distribution method impractical for many information retrieval tasks.

4.2 *Inverted file Replication*

Replication is the duplication of inverted file data on local disks of a parallel computer. Each node in the system has access to all the data locally, therefore the need to transmit large amounts of data is greatly reduced. The advantage of replication as against *On-the-fly* distribution is that the high communication cost is much reduced without loss of flexibility. The disadvantage is that space costs are considerably higher than any of the other distribution methods discussed here. Consider the following example;

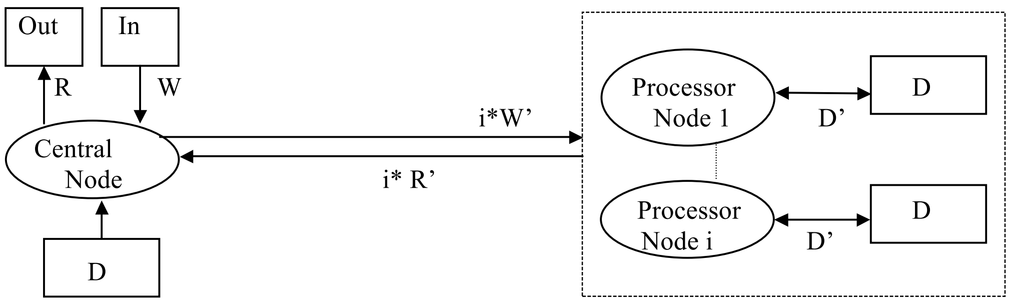

Fig 3: Example of abstract task on inverted file replication.

The key issue here is that all nodes have access to matrix D locally. In our abstract task we send each processor node W' together with some scheme for partitioning the matrix as required by the given computation or task. We then produce R' for each processor node and return the result back to the central node to compute the final R as would be done with *On-the-fly* distribution. A further advantage in having the whole matrix D available to the node is that load can be re-balanced by exchanging subsets of W' between the processor nodes, without having to communicate any aspect of D.

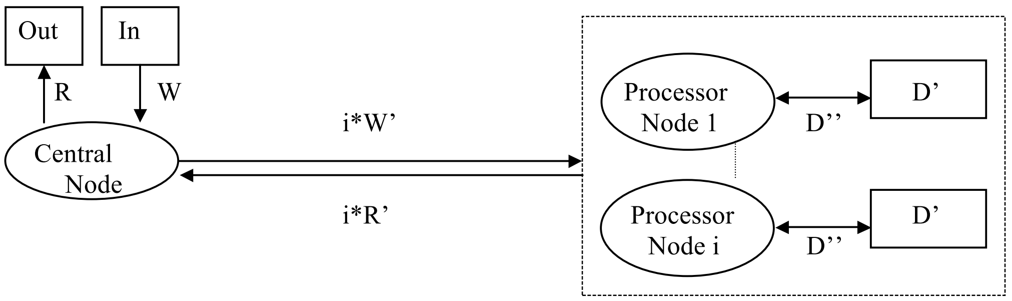

Fig 4: Example of abstract task on inverted file partitioning.

4.3 *Inverted file partitioning*

Partitioning is the fragmentation of inverted file data over local disks in a parallel computer (see fig 4). In the example given in figure 4 each node has access to its own subset of D, D' and which can only be accessed by that node. In terms of the abstract task, each node manipulates a subset of D', D'' in order to service work W or W'. The node services W or W' depending on the task and partitioning type. The advantage of partitioning is that the space costs are lower than *Replication*, but it is a static distribution method and is therefore not as flexible as the *On-the-fly* distribution method. The process of distribution is also much more complex than inverted file *Replication*.

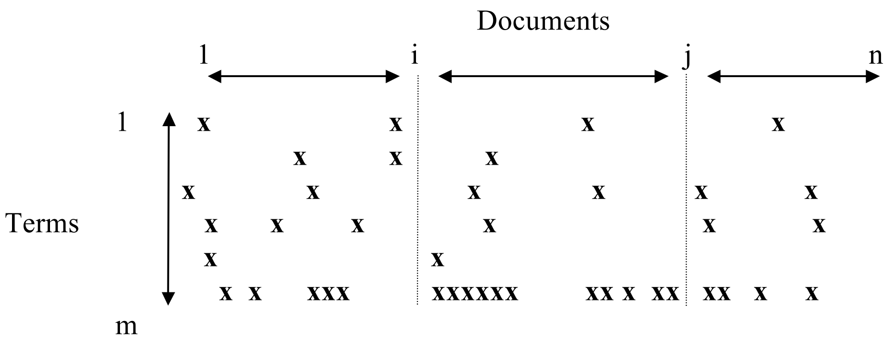

Fig 5: Example *DocId* matrix partitioning

There are two main inverted file *Partitioning* methods (Jeong and Omiecinski, 1995): by term identifier (*TermId)* and by document identifier (*DocId*). These partitioning methods are orthogonal to each other. With *DocId* partitioning the terms for a single document are placed on one disk, therefore postings for the same term may be held on multiple disks (see figure 5).

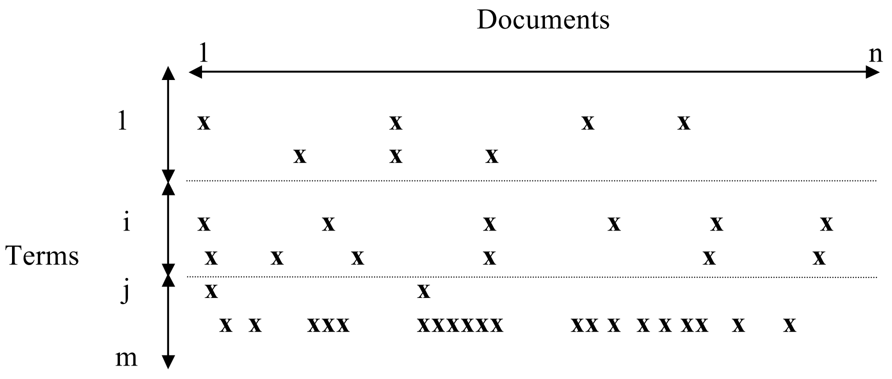

Fig 6: Example *TermId* matrix partitioning

We assume for arguments sake that we have three partitions in our example. In the example documents 1 to i-1 are given to node 1, documents i to j-1 are given to node 2 and documents j to n are given to node 3. The demarcation of the partitions is signified by the dotted line. With respect to our abstract task, we need to distribute W (if W is a query) to all partitions as all may have data for any or all of the terms in W. With *TermId* partitioning however, all postings for a given term are on one disk, therefore postings for the same document may be on multiple disks (see figure 6). In the example terms 1 to i-1 are given to node 1, terms i to j-1 are given to node 2 and terms j to n are given to node 3. With respect to our abstract task and our specific example given above on *DocId*, nodes get there own unique subset of work W' as each nodes has its set of unique terms.

5. Tasks in Information Retrieval

To recap, we define a task as being a specific aspect of an IR system that has its own functionality. We do not attempt to examine every task in IR, as the field is large. Thesaurus construction, clustering and hypertext creation tasks are among the notable exceptions that we do not investigate. We largely concentrate on what we regard as the main tasks in IR such as indexing and search: we define a main task as one with which an IR system using inverted files would be unable to function if such did not exist. The other (non-main) tasks studied can be built using the core tasks and extended as required. For example index update can be built from search and index functionality, while routing/filtering and passage retrieval can be built from search functionality. Each of the tasks to be studied in this chapter is defined below.

5.1 *The indexing task*

The indexing task is the process of taking raw text and building an index over that text using some criteria such as removal of stop words and stemming, etc. The process of indexing is one of the most computationally intensive aspects of IR requiring vast CPU, memory and disk resources (but is done only once and incremented thereafter). It is therefore a prime candidate for the application of parallelism. We consider *Partitioning* only for this task. *On-the-fly* distribution is irrelevant (our consideration of distribution is on inverted file data not raw text). Indexing on one processor and copying the data to the nodes can produce *Replicated* indexes.

5.2 *The probabilistic search task*

The probabilistic search task is the process of servicing queries on inverted files to produce a ranked list of documents using a term weighting scheme such as that derived by Robertson and Sparck Jones (1976). Query processing in this context is very fast, which is why inverted files in some form have become the dominant storage technique in IR. However, it is still possible to increase the speed of query processing further using parallelism, and by doing so increase the system throughput. With respect to distribution methods, *Replication* is not a suitable method for probabilistic search (with the possible exception of concurrent query service) while *On-the-fly* would restrict efficiency due to the extra communication overhead. It is likely that this extra overhead would outweigh any gain made by parallelism. Our discussion on the probabilistic search task is restricted to *Partitioning* methods only.

5.3 *The passage retrieval task*

The passage retrieval task described here is one used in Okapi experiments conducted within the TREC conference framework, in particular Okapi at TREC-3 Robertson et al (1995). We classify passage retrieval as being the retrieval of part of a document that is most likely to be of interest to a user, given a query. This algorithm takes an atom of text, say a paragraph, and iterates through the atoms of a document to a given maximum passage size. Passages that do not have query terms at either the start or end text atoms are ignored. Each passage is assigned a weight (using probabilistic model techniques), and the best-weighted passage used for display to the user, as a surrogate for the whole document or in relevance feedback. This passage processing technique is very computationally intensive and benefits from parallelism MacFarlane et al (2004). We study *Partitioning* methods only for the same reasons as given for the probabilistic search task.

5.4 *The routing/filtering task*

The idea behind information filtering is to disseminate incoming documents to users who require them. Users have long term information needs that may be satisfied by newly published documents. A method for information filtering is to take the documents that have been marked relevant by the user in the past and apply a relevance feedback mechanism to obtain a set of terms that can be applied to the new documents. We use TREC definitions of routing and filtering Harman (1996). In routing we provide the user with the top *n* documents, while in filtering we make a binary decision on which *n* documents will be presented to the user. There are a number of filtering techniques: batch filtering that takes *n* documents as a batch and adaptive filtering where documents are considered one at a time, with the possibility of feedback after each one. In Robertson et al (1995) 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". This is a combinatorial optimisation problem in term selection. A number of Hill Climbers have been implemented in Okapi at TREC-4 Robertson et al (1996) to solve this problem. The method we concentrate on in this chapter is the 'Find Best' algorithm which is a steepest ascent Hill Climber – the best term is chosen from on the terms in the evaluation set until some stopping criteria has been reached e.g. no further improvement can be made. In each algorithm a number of operations are available for term selection: add only, remove only and add/remove. Query terms can also be re-weighted any number of times. We consider add only with no re-weighting, and also with re-weighting terms twice. It is this term selection task in routing/filtering that we attempt to model here. We examine all the data distribution techniques theoretically for this task.

5.5 *The index update task*

The index update task consists of a number of different aspects. We consider the issue of transaction processing where a transaction is either a document to be inserted or a probabilistic search request. We define index update as data to be periodically added to the index when a buffer with document insertions has exceeded some memory limit. This requires some form of index reorganisation which must be done concurrently with transaction processing to prevent delays: we assume that transaction processing cannot be suspended and specify a requirement that queries should be serviced as soon as possible. It should be noted that we consider insertions only, not deletions or modifications. To do otherwise would complicate the modelling process further and in any case most text collections are archival in nature (the web being a notable exception). Index maintenance is a computationally intensive activity and we study the modelling of the task in order to investigate the viability of parallelism in a transaction processing context. We study partitioning methods only for the same reasons as given for the probabilistic search task.

6 A synthetic model of performance for parallel IR

In this section we briefly outline synthetic models of performance in order to compare distribution methods for all tasks under consideration in the chapter (a detailed description of the actual models themselves can be found in MacFarlane (2000)). However, due to practicalities we do not study absolute performance of the tasks (for reasons stated above), and our emphasis is restricted to the derivation of synthetic models that can only be used for comparative purposes. Not having to address the issue of absolute performance simplifies the process of modelling greatly. While our primary aim is to produce models that are strong enough to compare distribution methods and make choices between them, we also try to look beyond this simple requirement. We would like models that are good enough to predict the relative difference between data distribution schemes. We would also like to be able to make generic statements about parallel IR performance beyond the algorithms and architectures which we examine in this chapter: this may be difficult to do given the range of systems described in the literature (see Rasmussen (1992) and MacFarlane et al (1997)). These issues will be examined later on in the chapter by reviewing the empirical results gathered. The algorithms and methods modelled in this chapter are those described in MacFarlane et al (1999). We have a number of general variables for the models that are declared in table 1.

T_{cpu}	:	CPU time (for some operation).
$T_{i/o}[x]$	:	I/O time - Components: $1T_{seek} + x\ T_{trans;}$
		T_{seek} : Time to seek for I/O
		T_{trans} : Time to transfer data for I/O
T_{comm}	:	Communication time
T_t	:	Unit Time
P	:	No of nodes in a parallel machine
LI[P]	:	Load imbalance estimate at P processors

Table 1. General variables for the synthetic models

The format of the models is functional. This allows us to specify equations and reuse them in other defined equations. This makes it easier to replace various aspects of a given model in order to study different type of methods not under consideration in this chapter such as query processing optimisation and compression. We attempt to make our models as generic as possible. All functions return a single figure in abstract time. The functions only take variables as arguments: we do not specify higher order functions. Lookup values declared in the form x[y] (e.g. LI[P]) are not recorded as parameters but global variables. The scope rules for any declared variable are the normal ones found in most programming languages: variables declared locally take precedence over global ones. Sequential and parallel models are declared for all tasks. Simplifying assumptions for each of the models is declared in the relevant sections.

We make a number of assumptions in the general variables that impact (with varying degrees) on the synthetic models. We assume a low latency network in order to simplify the modelling of communication (otherwise we would have to break down T_{comm} using a $T_{comm}[x]$ format). For I/O we do allow two forms as blocks of data can be either static or dynamic. The Ti/o form of the variable can be used if fixed size blocks are transferred, and it is safe to assume that the balance between transfer and seek time is constant. We use the Ti/o[x] where variable sized blocks are transferred and the balance between seek and transfer time must be an integral part of the modelling process. For seek time we assume that an I/O request entails a single disk head movement. We assume an accumulated increase in load imbalance (variable LI[P]), at a rate of 0.015 for all synthetic models. It is difficult to know what the load balance will be for the parallel version of a particular task, without running a program and measuring the imbalance. We take this approach to provide a reasonable level of load imbalance for a given parallel machine size. For a given model we assume that the same parallel machine is used, that is the communication, CPU and I/O costs are identical across nodes in the machine: we do not address the issue of heterogeneous parallelism in the models (e.g. nodes with different architectures).

What follows in section 7 is a brief description of the model is given for each task described in section 5, and the results using that model are examined and compared with empirical data. Details of the models and how they were constructed and derived can be found on a web site from MacFarlane (2000).

7 Empirical examination of synthetic model

Our purpose in this section is twofold, to compare the theoretical results in order to show which distribution method is appropriate for each task, and then to compare the theoretical results with empirical results from our implementation of the tasks in order to see how well the models can distinguish different distribution methods. In each section we describe the evidence used to instantiate the theoretical models, in order to produce them. All diagrams declare the theoretical results on abstract unit time as against the number of processors (P).

7.1 *Comparative results using indexing models*

The evidence we used to develop the theoretical models is from the BASE1 collection Hawking et al (1999): this collection consists of 187,000 documents with an average document length of around 465 words. We use the following values in the models: T_{cpu} of 0.01, $T_{i/o}$ of 0.015 and T_{comm} of 1 – these were chosen to reflect he approximate balance between the different aspects. We assume the transfer to and from disk is with fixed sized blocks. With communication time we assume a fast network, given the amount of data to be transferred between nodes. We produce three theoretical models, one for the *TermId* partitioning method and two for the *DocId* partitioning method. The two models for the latter partitioning scheme reflect different types of builds: *Local* where documents are saved to and indexed from a local disk and *Distributed* where a single master processor distributes documents to other processors in the parallel computer.

From figure 7 it can be seen that there is an advantage in theory in using *DocId* partitioning over the *TermId* method for indexing in that the models for the former predict better performance over the latter on all parallel machine sizes. It should be noted that the model predicts a narrowing of the gap between partitioning methods with increasing machine size (we will examine this issue later in this section). There is little difference between *Local* and *Distributed* builds in *DocId*, but we have assumed a high bandwidth network in the instantiated models (the graph for *Local* build *DocId* is obstructed by *Distributed* build *DocId* as there is virtually no difference in theoretical time between them). If we assumed a much lower bandwidth network, there would be a clear difference between the builds and *TermId* would not compare well with either of the *DocId* methods.

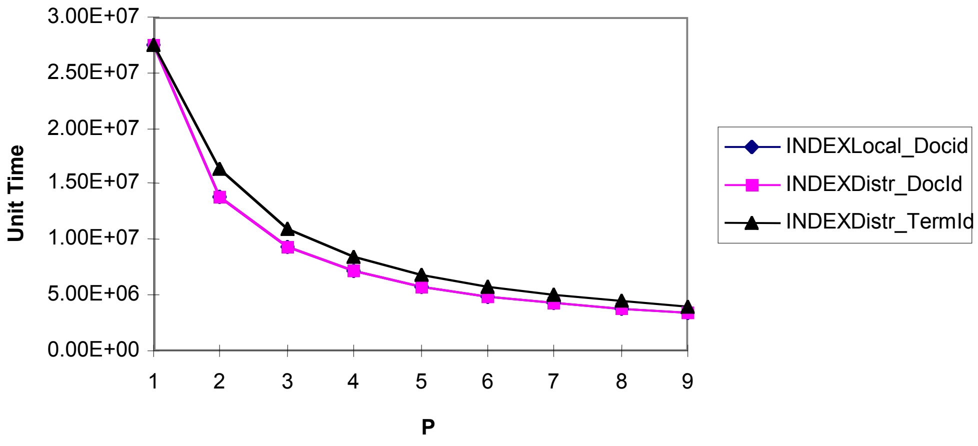

Fig 7. Synthetic indexing performance on 1 to 9 Leaf nodes

When comparing the performance of both partitioning methods with our empirical results MacFarlane at al (2005), the synthetic model for indexing was able to predict that *DocId* builds are faster than *TermId* builds. Our empirical results show definitively that the *DocId* partitioning scheme is a superior method in terms of speed, than the *TermId* method MacFarlane et al (2005). The clear reason for this was the extra communication costs, implied by the *TermId* method: even using the assumption of a fast network. The extra communication costs are largely due to an extra merging process needed for *TermId* indexing. Documents are distributed to processors in *TermId* for a parsing phase, which meant that intermediate results must be exchanged, requiring N*(N-1) data sets to be transmitted over the network. Problems in the modelling of communication in the synthetic model on this merging process meant that a widening gap in performance between the two partitioning methods for increasing parallel machine size was not anticipated (the model actually predicted a narrowing of the gap in run time between the two partitioning methods). This failure in modelling is discussed in more detail in the conclusion. With respect to *Local* versus *Distributed* builds in *DocId* partitioning, no direct comparison was made, as it was clear that *Local* build indexing run time would be faster.

7.2 *Comparative results using search models*

The BASE1 collection was again used as source of evidence to develop the theoretical search models. The same values for T_{cpu} and T_{comm} where used as in the indexing models, but assumed in the search case that variable sized blocks would be transferred from disk and therefore split $T_{i/o}$ into T_{trans} and T_{seek} using the values 0.015 and 0.1 respectively. These are the values we assumed for all the theoretical search models discussed below. We assumed a query size of 2.5 terms for the models as users tend to submit just over two terms per query (Silverstein et al, 1999). The results of applying these values shown in fig 8 demonstrate that in theory the *DocId* partitioning method ($SEARCH_{docid}$) would perform better than the *TermId* method using either a parallel sort ($SEARCH_{termid2}$) or sequential sort ($SEARCH_{termid1}$). The comparative results also predict that the *TermId* method with parallel sort will outperform the algorithm with a sequential sort by a substantial amount: the synthetic model predicts that a sequential sort will be a bottleneck. The prediction on the *TermId* partitioning method with a sequential sort is particularly bleak, with little or no advantage to be gained from parallelism.

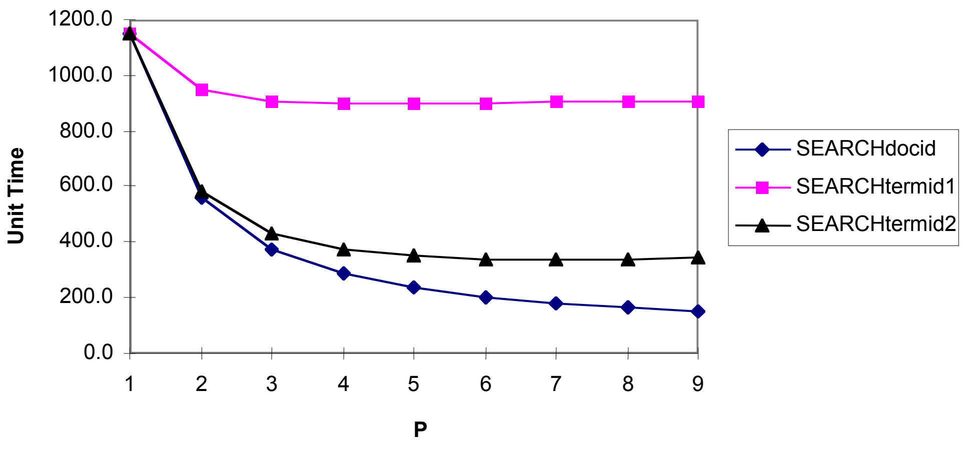

Fig 8. Comparative results for search models on 1-9 leaf nodes

Our empirical results MacFarlane et al (2000) show that runs on the *DocId* partitioning scheme outperform a *TermId* scheme, whether or not a parallel sort is implemented. The prediction in the synthetic model that a sequential sort would be a bottleneck is confirmed by empirical results (MacFarlane et al, 2000). The synthetic model is also able to predict the relative performance difference between the partitioning methods to a great extent. However our empirical results show that *TermId* with parallel sort performance is nearer to *TermId* with sequential sort, whereas our synthetic model predicted that search on *TermId* with parallel sort would be nearer to *DocId*. The implemented parallel sort, required that the final merged results be distributed to processors and the sorted data retrieved when each processor has completed its sort: as with indexing this communication was not dealt with correctly by the synthetic model.

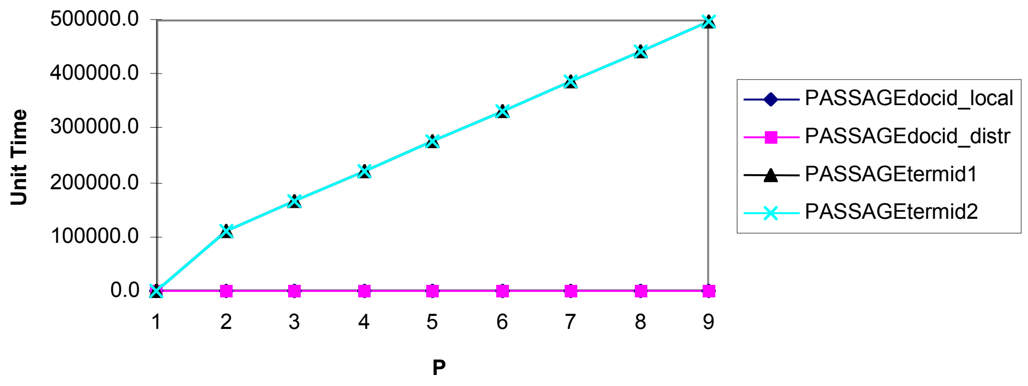

Fig 9. Synthetic passage retrieval model on 1 to 9 leaf nodes

7.3 *Comparative results using passage retrieval models*

We used the same values for the models as per the search models in section 7.2 above, but need some further assumptions to make on the average number of text segments to inspect per document and the total number of documents on which to do passage retrieval: we chose the values of 11 and 1000 respectively. Figure 9 shows the results on the theoretical passage retrieval models for both partitioning methods. There is clearly a significant problem with using *TermId* partitioning with the passage retrieval algorithm we model. The predicted comparative performance is so poor that the *DocId* models appear to be near zero. It is clear from the model that predicted communication costs would increase the run time of any parallel program using such a partitioning method. Given the problems with modelling communication in both indexing and search models, the costs for passage retrieval are likely to a significant underestimate. As a result we chose not to run any practical experiments using this partitioning method, and therefore concentrated on the *DocId* partition method alone using both the *local* and *distributed* document allocation methods. With the *local* method, the local top 1000/P documents are used for passage processing, whereas in the *distributed* method the global top 1000 documents are processed.

Fig 10 shows synthetic model results for *DocId* partitioning only. With both types of passage retrieval using *DocId* the prediction is time reduction with increasing numbers of processors, but the local method (marked $\text{PASSAGE}_{\text{docid\_local}}$) shows slightly better theoretical results than the distributed method (marked $\text{PASSAGE}_{\text{docid\_distr}}$). There is a little extra communication for the *distributed* method, where the master process needs to identify the top 1000 documents, and inform the search processes as to which of their documents are to be examined for passages. Also, there is no guarantee that passages will be equally distributed in the *distributed* method, and this has the potential to effect load balance. This aspect is very difficult to model, as the distribution of documents to which passage processing is to be applied, cannot be determined prior to search.

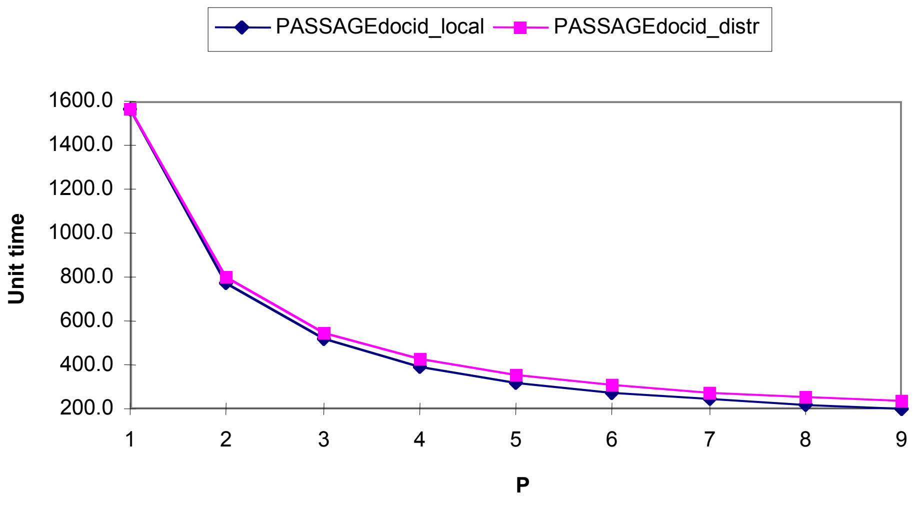

Fig 10. Synthetic passage retrieval models - *DocId* only

The empirical results show that our synthetic model correctly predicted that the *local* passage processing method would outperform the *distributed* version. However, the model was unable to predict the relative difference due to the super linear speedup performance for the former and the erratic nature of the performance in the latter on short queries (MacFarlane et al, 2004). The remarkable performance of the *local* passage processing method was somewhat hard to fathom at first, as the number of passages processed was around the same as the *distributed* method. However individual text atoms can vary in size considerably, it is clear that the local method was processing a different set of documents which were less computationally costly to process than the *distributed* method. The variable performance of the *distributed* method was due to the different numbers of documents processed at each machine size, which tended to vary quite considerably and non deterministically. The factors found by empirical experiment added yet another unforseen level of complexity to the modelling process which we are currently unable to deal with, that is modelling the effect of text atoms on passage processing.

7.4 *Comparative results using term selection models*

For our modelling of the term selection we assumed that the optimisation process would be done on 300 terms with a maximum of 100 iterations. The values chosen for these variables were used in the empirical experiments (MacFarlane et al, 2003). We modelled only one of the hill climbers under consideration, namely 'Find Best'. Fig 11 shows the comparison between models on all distribution and partitioning methods with the number of processors (P) set between 10-100.

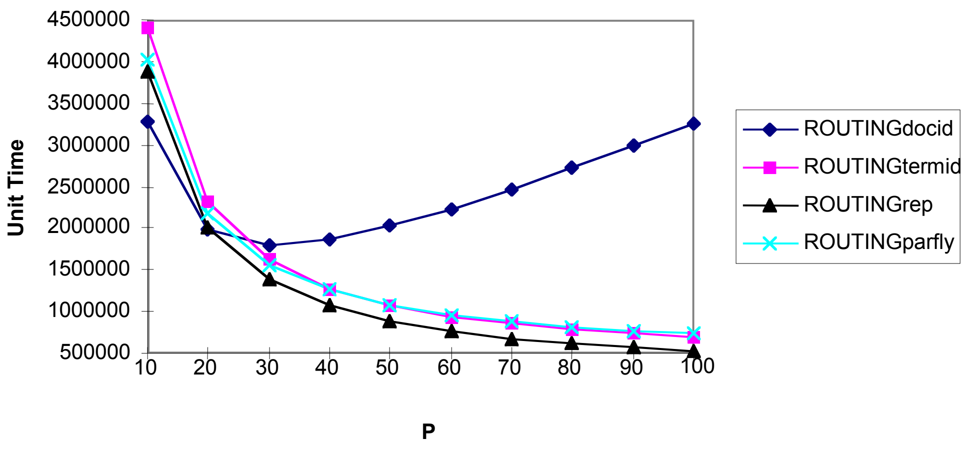

Fig 11. Term selection model results using large slave node set and iteration sizes

The theoretical models predict that the best performing distribution scheme overall would be the *replication* method. On smaller processor sets, the *DocId* partitioning method shows better theoretical results, but predicts that the performance would deteriorate substantially due to the restriction on the level of parallelism in that distribution method: the communication/ computation balance would be skewed with increasing numbers of processors. We therefore did not implement the *DocId* partitioning method, as it was clear that other distribution methods where more likely to succeed when larger parallel machines were used. The prediction with respect to *TermId* partitioning and *On-the-fly* distribution is that there is little difference between them particularly with large processor sets.

The synthetic model was successfully able predict that *Replication* is a superior method to *On-the-fly* distribution, but not that the latter would perform so poorly (MacFarlane, 2000). The 'Find Best' algorithm show near linear speedup on the *Replication* method. However, the results for the *On-the-fly* distribution method actually show a slowdown with more processors than just using one processor: the performance gets worse with more processors. This is in spite of the fact that we used a 50 Mb per second network or on a large scale Fujitsu parallel computing (with 100 processors). The bottleneck at the synchronisation point, where data must be exchanged between iterations proved to the stumbling block for the method.

We can make a further statement on the synthetic models, given the evidence found with *On-the-fly* distribution. The *TermId* partitioning method would require more communication at the synchronisation point than *On-the-fly* distribution. There is no guarantee that query terms with be distributed evenly with the *TermId* method, and this has the potential to effect load balance detrimentally. The *TermId* partitioning method is therefore not a viable method, and as a result we did not implement this scheme for our experiments.

7.5 *Comparative results using index update model*

In the index update model we assumed that for every 10 searches, there would be 1 update to process. Fig 11 shows the prediction using these values and compares the theoretical performance between both types of partitioning methods where transactions are affected by a concurrent index update (models are labelled with the suffix RO) and normal transaction processing without contention for resources. What we mean by concurrent index update is that updates held in a temporary buffer are merged with the inverted file, while transactions are serviced concurrently.

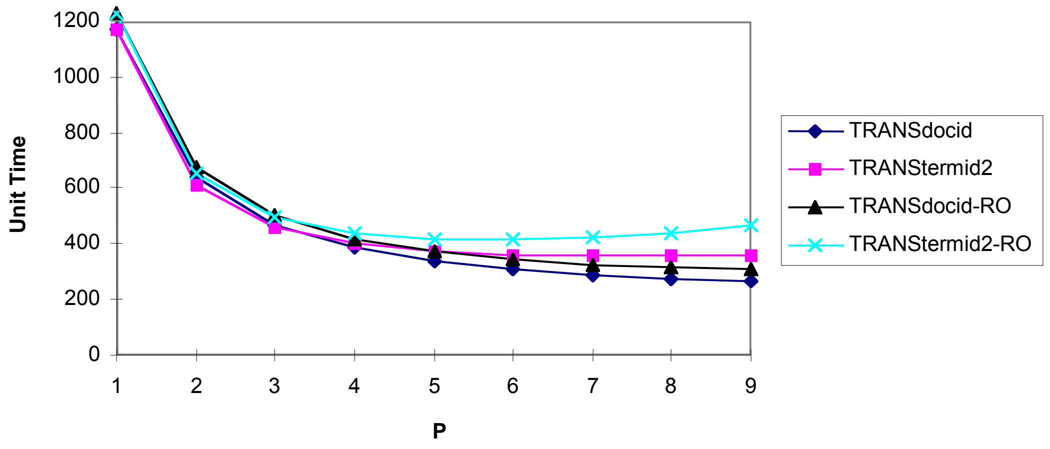

Fig 11. Comparison between partitioning methods in presence and absence of index update

The models predict that *DocId* partitioning will outperform *TermId* whether or not there is contention for resources due to index update. As you would expect, where there is contention for resources the predicted performance is worse than non-contention models. The model on *TermId* partitioning in the presence of an index update predicts a deterioration in performance with increasing numbers of processors: with *DocId* the prediction is that performance will remain constant after a certain number of processors is reached.

Our synthetic model for the index update task is strong enough to distinguish between the partitioning methods (MacFarlane et al, 2007). Problems highlighted in our probabilistic search experiments (see section 7.2 above) impose severe restrictions on transaction processing when the *TermId* method is used, which are difficult to solve within our experimental context. These problems (most notably the sort aspect of search) had an impact on the relative difference between the two partitioning methods during transaction processing, a problem the synthetic model was not able to deal with very well. The synthetic model correctly predicted that the performance of transactions serviced on *TermId* indexes during an index reorganisation would deteriorate (although the empirical results were relatively worse because of a list size problem described in (MacFarlane et al, 2006). The synthetic model also correctly predicted that transaction performance on *DocId* partitioning indexes would be constant after a given number of leaf nodes is reached in the same situation.

7. Conclusions and Further research

The synthetic models outlined in this chapter predict that for most tasks the *DocId* partitioning method would be the better performing data distribution scheme of those studied. For the index, probabilistic search, passage retrieval and index update tasks the prediction is unambiguous. For the passage retrieval task in particular it is very clear that the *TermId* partitioning method is simply not viable: we did not therefore implement this partitioning method for the passage processing task (MacFarlane et al, 2000). The theoretical evidence on the routing/filtering task is more complicated and therefore needs more discussion. On small numbers of processors the prediction is that *DocId* partitioning would be the best data distribution scheme, but on larger parallel machines the performance would deteriorate due to excessive communication. Due to the restrictions on inter-set parallelism, the proposed method for *DocId* partitioning does not show the same promise as intra-set parallelism usable on the other distribution methods. Of the other three, the prediction is that *replication* would be the best performing method and that performance with *on-the-fly* distribution would be about the same with *TermId* partitioning. We therefore concentrated our efforts on *replication* and *on-the-fly* distribution schemes, partly because of the theoretical comparison and partly because such methods are easier to manage. Only one indexing run needed to be initiated in order to undertake those experiments, whereas with *TermId* partitioning a new indexing run would be required for every processor added.

Further work on the synthetic modelling technique used in this chapter is merited both in terms of strengthening the actual model and extending it for use in other tasks not studied here. While the models were able to correctly predict that one distribution scheme was superior to the others, given the empirical evidence, they were not able to predict the relative difference between the models. This was largely because the synthetic models were not able to model communication completely. The key aspect to concentrate on initially therefore will be the modelling of communication, given that it was such a significant problem in our models. The problems in being able to do this successfully should not be underestimated. The model will not only have to cope with interactions between two processors, it will also have to model the pattern of communication throughout the whole system (one of the reasons for our simplified modelling of communication was to avoid this complexity). A particular problem with passage retrieval, is the ability to model the effect of the text atoms on passage processing: this non-deterministic factor is very hard to model. An interesting and worthwhile piece of research would be to extend the functional modelling and use formal techniques to prove various aspects of it: this may well provide insights that would not be obtained otherwise. When many problems have been solved by using synthetic modelling, and more of an understanding of the theory of performance of parallelism in IR is obtained, it may well be possible to derive an analytical model of performance in order to model real performance. Hopefully, this analytical model would be able to predict the actual performance in a given IR task, more accurately then we can at present.

References

Cardenas, A.F. (1975). Analysis and performance of inverted data base structures. *Communications of the ACM*, 18 (5), 253-263.

Clarke, C., Craswell, N. and Soboroff, I. (2005). Overview of the TREC 2004 Terabyte Track, in *Proceedings of the Eleventh Text REtrieval Conference, (TREC 2004)*, (eds E. Voorhees and L. Buckland), NIST Special Publication 500-261, Gaithersburg, U.S.A, [available on: http://trec.nist.gov - visited 15th March 2007].

Fedorowicz, J. (1987). Database performance evaluation in an indexed file environment. *ACM Transactions on Database Systems*, 12 (1):85-110.

Fox, E., Betrabet, S., Koushik, M., and Lee, W. (1992). Extended Boolean models in *Information Retrieval, Data Structures and Algorithms*. (eds W.B. Frakes, and R. Baeza-Yates), Prentice-Hall, N.J., pp393-418.

Harman, D.K. Fox, E. Baeza-Yates, R and Lee, W. (1992). Inverted Files, in *Information Retrieval, Data Structures and Algorithms*. (eds W.B. Frakes, and R. Baeza-Yates), Prentice-Hall, N.J., 28-43.

Harman, D.K. (1996). Overview of the fourth text retrieval conference (TREC-4), in, *Proceedings of the Fourth Text Retrieval Conference (TREC-4)*, (ed D.K. Harman) NIST Special Publication 500-236, Gaithersburg, U.S.A, 1-24.

Hawking, D. (1996). Document retrieval performance on parallel systems. In: ARABNIAL, H.R., ed, Proceedings of the 1996 International Conference on Parallel and Distributed Processing Techniques and Applications, Sunnyvale, California, August 1996, (Athens: CSREA): 1354-1365.

Hawking, D., Craswell, N. and Thistlewaite, P. (1999). Overview of TREC-7 very large collection track. In: Voorhees, E.M. and Harman, D.K., eds. Proceedings of Seventh Text Retrieval Conference (TREC-7), Gaithersburg, USA, November 1998. NIST SP 500-242, (Gaithersburg: NIST): 257-268.

Jeong, B., and Omiecinski, E. (1995). Inverted file partitioning schemes in multiple disk systems. IEEE Transactions on Parallel and Distributed Systems, 6 (2): 142-153.

MacFarlane, A. Robertson , S.E. AND McCann, J. A. (1997). Parallel computing in information retrieval – an updated review. Journal of Documentation 53(3), 274-315.

MacFarlane, A., McCann, J. A. and Robertson, S.E. (1999). PLIERS: a parallel information retrieval system using MPI. In: Dongarra, J., Luque, E. and Margalef, T., eds. Proceedings of 6th European PVM/MPI Users' Group Meeting, Barcelona, Lecture Notes in Computer Science 1697, (Berlin: Springer-Verlag): 317-324.

MacFarlane, A. (2000). Distributed Inverted files and performance: a study of parallelism and data distribution methods in IR, PhD thesis, City University London, August 2000. [Available on: http://www.soi.city.ac.uk/~andym/PHD/: visited 7th November 2005]

MacFarlane, A., McCann, J. A. and Robertson, S.E. (2000). Parallel search using partitioned inverted files. In: DE LA FUENTE, P., ed, Proceedings of String Processing and Information Retrieval - SPIRE 2000, September 2000, A Coruna, Spain, (Los Alamitos:IEEE Computer Society Press), 209-220.

MacFarlane, A. Robertson , S.E. and McCann, J. A. (2003). Parallel computing for term selection in routing/filtering, In: Sebastiani, F. Proceedings of ECIR 2003, LNCS 2633, 537-545.

MacFarlane, A. Robertson , S.E. and McCann, J. A. (2004). Parallel computing for Passage Retrieval. ASLIB Proceedings: New Information Perspectives, 56(4), 201-211.

MacFarlane, A., McCann, J. A. and Robertson, S.E. (2005). Parallel methods for the generation of partitioned inverted files. ASLIB Proceedings: New Information Perspectives, 57(5), 434-459.

MacFarlane, A., McCann, J. A. and Robertson, S.E. (2007). Parallel methods for the update of partitioned inverted files. ASLIB Proceedings: New Information Perspectives (to appear).

Rasmussen, E. (1992). Parallel information processing. In: Williams M.E., Annual Review of Information Science and Technology (ARIST), Volume 27, 99-130.

Ribeiro-Neto, B., Moura, E.S., Neubert, M.S., and Ziviani, N. (1999). Efficient distributed algorithms to build inverted files. In: Hearst, M., Gey, F. and Tong, R., Proceedings for the 22nd International Conference on the Research and Development in Information Retrieval, SIGIR'99, (New York: ACM Press): 105-112.

Robertson, S.E., & Sparck Jones, K. (1976). Relevance weighting of search terms. JASIS, May-June: 129-145.

Robertson, S.E., Walker, S., Jones, S., Hancock-Beaulieu, M.M. & Gatford, M. (1995). Okapi at TREC-3. In: Harman, D.K., ed. Proceedings of Third Text Retrieval Conference, Gaithersburg, USA, November 1994, NIST SP 500-226, (Gaithersburg: NIST): 109-126.

Robertson, S.E., Walker, S., Jones, S., Hancock-Beaulieu, M.M., Gatford, M. and Payne, A. (1996). Okapi at TREC-4. In: Harman, D.K., ed, Proceedings of the Fourth Text Retrieval Conference, Gaithersburg, U.S.A, November 1995, NIST SP 500-236, (Gaithersburg: NIST): 73-96.

Salton, G., and Buckley, C. (1988). Parallel text search methods. Communications of the ACM, 31 (2): 202-215.

Silverstein, C., Henzinger, M., Marais, H, and Moricz, M. (1999). Analysis of a very large web search engine log. SIGIR Forum, 33 (1): 6-12.

Stanfill, C., Thau, R., and Waltz, D. (1989). A parallel Indexed algorithm for Information Retrieval. In: Belkin, N.J., and van Rijsbergen, C.J., eds. Proceedings of the 12th annual conference on research and development in Information Retrieval, SIGIR'89, (New York: ACM Press): 88-97.

Stone, H. S. (1987). Parallel querying of large database: a case study. IEEE Computer, 20 (10): 11-21.

Tomasic, A., and Garcia-Molina, H. (1993a). Performance of inverted indices in shared-nothing distributed text document information retrieval systems. Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems, (Los Alomitos: IEEE Computer society press): 8-17.

Tomasic, A., and Garcia-Molina, H.. (1993b). Caching and database scaling in distributed shared-nothing information retrieval systems. In: Buneman, P., and Jajoida, S., eds, Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. (N.Y.: ACM Press): 129-138.

Wolfram, D. (1992a). Applying informetric characteristics of databases for IR system file design, part i: informetric models. Information Processing and Management, 28 (1): 121-133.

Wolfram, D. (1992b). Applying informetric characteristics of databases for IR system file design, part ii: simulation comparisons. Information Processing and Management, 28 (1): 135-151.