THE AUSTRALIAN NATIONAL UNIVERSITY

TR-CS-96-04

A Parallel Architecture for Query Processing Over A Terabyte of Text

Peter Bailey and David Hawking

June 1996

Joint Computer Science Technical Report Series

Department of Computer Science
Faculty of Engineering and Information Technology

Computer Sciences Laboratory
Research School of Information Sciences and Engineering

A Parallel Architecture for Query Processing Over A Terabyte of Text

Peter Bailey and David Hawking[*]
Cooperative Research Centre For Advanced Computational Systems
Department of Computer Science
The Australian National University
Canberra ACT 0200 AUSTRALIA
{dave,peterb}@cs.anu.edu.au

June 7, 1996

[*]Equal joint authors.

Abstract

The Parallel Document Retrieval Engine (PADRE) has previously demonstrated that full text scanning methods supported by parallel hardware permit powerful query constructors and rapid response to changing document collections. Extensions to PADRE have been designed and implemented which make use of parallel secondary storage to allow each procesing node to handle data up to 32 times the size of its primary memory. Using the largest purchasable machine on which PADRE currently runs, these increase the maximum possible collection size to one terabyte. This paper addresses the practicality of achieving this limit and the extent to which the performance, responsiveness, functionality and scalability of the full text scanning PADRE are preserved in the extended version.

KEYWORDS Text retrieval, indexes, dictionaries, parallel computing.

1 Introduction

PADRE [8, 5] is a free text system designed to retrieve documents relevant to a specified research topic from among a large collection. There is no restriction on the type of material to be searched but typical applications might operate over collections of news reports, patents, legislation or technical documentation.

PADRE is implemented on Fujitsu AP1000 [9, 12] series machines, parallel, distributed memory multicomputers whose salient features are listed in table 1. The processing speed and memory capacity of these systems allows TREC [2] style research topics to be quickly processed over multi gigabyte collections, using full text scanning (that is, pattern matching operations over memory-resident raw text). The TREC collection, three gigabytes in all, has been used extensively in testing and measurement phases of the present work but precision-recall performance on the TREC task is not considered here.

PADRE exploits parallelism by distributing the document collection across the available processing nodes. Individual documents are not split. For the most part, each node can operate independently on its own subset of the data.

PADRE pattern matching operations produce match sets whose elements are byte pointers into the raw text. Match sets can be operated on with set and proximity operators to produce new match sets. Pairs of match sets can be used to dynamically define text components (such as authorname or abstract) to which subsequent searches may be restricted.

The computation of a match set normally causes the relevance estimates for affected documents to be updated. It is also possible to display the lexicographic context of each match or a sample of the matches.

Previous work on PADRE [3, 4, 8, 5, 7] has demonstrated that:

1. A 5 gigabyte collection can be easily handled on an existing AP1000 configuration, entirely in primary memory. This figure would extrapolate to over 40 gigabytes on the maximal AP1000+ system.

2. Access to the full raw text potentially allows retrieval to be based on any function computable over text. For example, PADRE queries may use any of the functionality of the GNU regular expression library.

3. Full text scanning also permits PADRE to respond almost instantly to changes in the document collection such as addition and deletion of documents.

	ANU AP1000	**Maximal AP1000+**
Memory	distributed	distributed
Memory per node	16Mbyte	64 Mbyte
CPUs	25 MHz SPARC 1+	50 MHz SuperSPARC
Number of nodes	128	512
Number of disks	32	512
Disk capacity(each)	0.5 gbyte	4 gbyte
Peak disk trasfer rate(each)	3 Mbyte/sec	11 Mbyte/sec
Addressing	32 bit	32 bit
Front-end	Sun workstation	Sun workstation
Communications	2-D torus plus broadcast	2-D torus plus broadcast
Comm.s speed	25 MByte/sec. per link	25 Mbyte/sec. per link

Table 1: Configuration details of the two Fujitsu configurations discussed in this paper. The ANU AP1000 is the machine on which all reported measurements were obtained and the maximal AP1000+ is the target host for the terabyte collection. Note that although a 1,024 node diskless AP1000 has been built, cabinetry constraints limit diskfull configurations to a maximum 512 nodes.

4. Document retrieval is an inherently parallel problem since the document collection can be divided into N approximately equal chunks, capable of being searched independently except for collection frequency calculation and global ranking.

5. Using full text scanning, PADRE's capacity and performance scales almost linearly with increasing number of processing nodes.

Some real world collections of text comprise hundreds of gigabytes of data and pose considerable probems for conventional retrieval systems running on sequential machines. The observed scalability of PADRE suggests that exploiting parallelism should provide an answer to these problems.

It is likely that, in the next few years, parallel machines will be built with in excess of a terabyte of main memory. Indeed, such configurations have already been designed. However, their cost is likely to exceed the budget of most document retrievers for the immediate future!

This paper describes an approach to increasing the amount of text which can be handled on a given memory configuration without seriously compromising the advantages of the full text scanning approach. We propose a design, describe a prototype implementation based on inverted file indexes and dictionaries kept on [parallel] secondary storage, report measurements of capacity and performance and attempt to show that a currently marketed AP1000+ configuration could be used to provide acceptable response time to retrieval queries over a terabyte collection. A terabyte is a meaningful yardstick because it is the same order of magnitude as the volume of text data represented by the books in a large University library.

To our knowledge, the only previously published work addressing free text retrieval over text collections of this size has been carried out by Stanfill and various collaborators at Thinking Machines Corporation. Stanfill and Thau [10] describe parallel retrieval algorithms for the Connection Machine CM2 which they claim are capable of processing queries over 8 terabytes of text. Their system calculated vector distances between queries and documents using a dictionary[1] and an inverted file index. These structures were partitioned to optimise performance on the CM2 architecture. Proximity operators were not supported.

The emphasis of their paper (and also a related paper by Stanfill, Thau and Waltz [11]) is very much on the scoring and ranking of documents. They did not demonstrate a solution to the mammoth problem of storing and indexing multiple terabytes of data. Indeed, Stanfill and Thau *"presume that the database has been indexed, and each document reduced to a set of structures..."*.

2 Proposed Architecture For Handling Very Large Textbases

2.1 Indexes And Dictionaries

With a view to maintaining reasonable query response times when supporting multiple time-sharing users, PADRE has always included the ability to build and use inverted file indexes, kept in primary memory along with the raw text. The form of index adopted is simply a list of byte pointers to the start of each indexed word. This list is sorted so that the words referenced are in lexicographic order. Search terms can thus be located using a binary search. Note that each processing node need only index and need only refer to documents in its own assigned *chunk* of the data.

Though simple, this index format is potentially rich enough to support all PADRE operations except for regular expressions and literal searches not anchored at word starts. The latter and a subset of the former may also be implemented with some coding difficulty and loss of speed. Unlike some less information-preserving index formats reported in the literature, proximity operations are supported.

The sort of pattern matching arising from real-world retrieval queries normally requires insensitivity to case. To support this, PADRE indexes were recently made case insensitive, adding a case conversion operation to the cost of entering each term occurrence in the index and greater complexity to the actual matching operation.

The PADRE approach to parallelism also works for indexing. Dividing the index building task into many independent pieces is a very successful strategy. Previous work [8] has reported that 2 gigabytes of TREC data may be indexed in about 90 seconds on a large AP1000 configuration. Searches for literal words using indexes in primary memory are two to three orders of magnitude faster than the same search using full text scanning. However, in this fully memory-resident model, use of an index reduces the amount of text which may be processed.

The scalability of index-based searching is not likely to be as good as that of full text scanning because of Amdahl's Law. In the full text scanning method, the highly parallel part of the algorithm (the searching) dominates fixed communication and set up costs. However, when searching is speeded up dramatically using indexing, the fixed costs are relatively more significant.

Extending PADRE's data handling capacity to the terabyte level is achieved by dividing the collection in two dimensions instead of one. The collection is first divided horizontally into sub-collections and then each sub-collection is divided vertically across the processing nodes. We refer to the piece of a sub-collection held on a single processor node as a *chunk*. Sub-collections would normally correspond to natural broad divisions in the data such as those in the TREC collection. Figure 1 shows the division of an illustrative collection.

Sub-collections are indexed independently in primary memory and then the indexes are written separately to parallel disk. This method imposes some size constraints on sub-collections

[1]called a data map

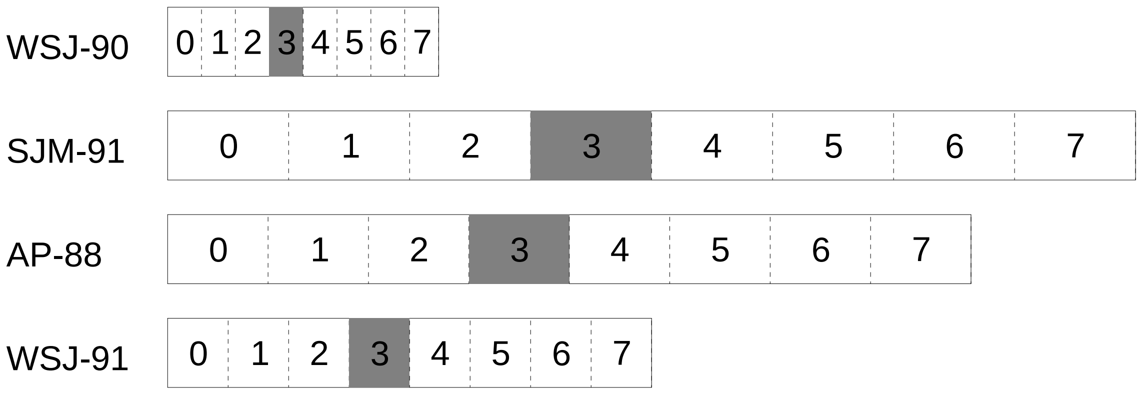


Figure 1: An illustrative collection (some of the TREC data) divided "horizontally" into sub-collections. Each of the sub-collections, whose relative size is indicated by the length of the rectangle, is shown distributed across eight processing nodes. No individual document is split across nodes. The part of a sub-collection assigned to a processing node is called a chunk. In this example, node 3 has been assigned the four chunks shaded in grey. (8 nodes)

but allows efficient index building and localised update when a collection changes.

So far, no attempt has been made to compress PADRE indexes, but advantage is of course taken of the horizontal and vertical division of data to reduce the number of bits needed for pointers. Experiments with compression techniques conducted by others [1] indicate that very considerable space savings may be achieved and that the extra processing cost for compression and decompression can be completely offset by the performance improvements due to reduced disk access times.

Quick access to the relevant section of an index on secondary storage is enabled by the use of a dictionary, kept in primary memory, which alphabetically lists all words encountered in an index, each with a pointer to the block of entries in the index which relate to it. The dictionary also records the size of each block, allowing rapid calculation of search term collection frequencies, without reference to disk.[2]

Remembering that, in our model, no processing node will ever be required to index more text than will fit in its local memory (up to 64 Mbytes on the AP1000+), it will be noted that a maximum of 26 bits are needed for the byte pointers in an index ($2^{26} = 64$ Mbyte). The same number of bits will also suffice to represent matchpoints within a chunk. However, in recording matchpoints it is necessary to identify the sub-collection in which they occurred. We have chosen to use the high-order six bits of our 32 bit matchpoint pointers to identify the sub-collection, allowing a maximum of $2^6 = 64$ sub-collections. Despite the inclusion of this additional information, PADRE set, component and proximity operators continue to work correctly as originally coded.

Assuming a perfectly even division of the collection into sub-collections, with the size of each chunk approximating half of a node's memory capacity, each processing node can handle $2^{25} \times 2^6 = 2^{31}$ bytes or 2 gigabytes of data. The present maximal AP1000+ configuration with disks comprises 512 ($= 2^9$) processing nodes, giving a collection limit of $2^9 \times 2^{31} = 2^{40}$ bytes or 1 terabyte.

In a PADRE index file, the high order six bits of the pointers do not need to identify the sub-collection. In the current implementation, these bits are used to reduce the amount of disk

[2]Global collection frequencies are computed using the hardware-assisted global reduction operators available on the Fujitsu AP1000.

I/O when processing queries: Three bits are used to encode the case mix of letters within the word (e.g. all lower case, all upper case, first letter capitalised etc), and three bits to encode the punctuation at the end of the word (e.g. fullstop, comma, parenthesis, etc). In each field, one value is reserved for uncommon cases which must be resolved by referring to the raw data. This technique ensures that the vast majority of query terms, such as used in processing TREC topics, can be identified using the index alone, without any recourse to the text itself.

Were PADRE to be ported to a different type of parallel hardware with more or less memory per node, different number of processing nodes or different pointer sizes, the tradeoffs underpinning the design decisions in this section would need to be revisited.

2.2 Super Dictionaries

In order to enable multiple sub-collections to be searched as a single collection, a data structure referred to as a *super dictionary* is created by merging the dictionaries for each sub-collection. The part of a super dictionary associated with a processing node comprises two parts. The first is a complete list of all unique words in the combined sub-collection chunks for the node. The second is a record that includes the frequency of occurrence of the word in each sub-collection chunk, and the relevant offsets into the sub-collection chunks' index files. A diagram of the super dictionary architecture is given in Figure 2.

For efficiency, PADRE keeps the first part of the super dictionary (the word list) in primary memory.

2.3 Document Information Structures

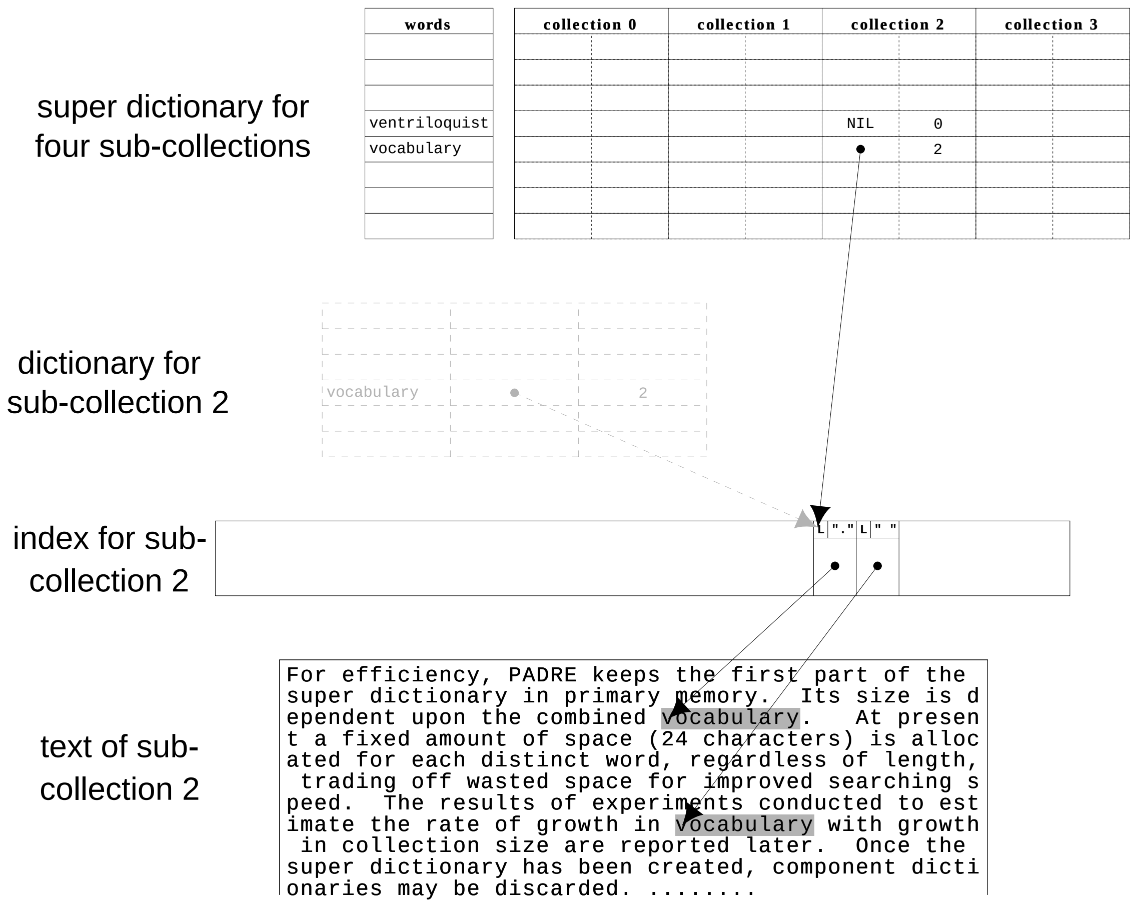
PADRE records various information about each document such as start point, accumulated positive relevance, accumulated negative relevance, maximum single contribution to relevance and various flags. At present this information totals 24 bytes. In the TREC collection, documents range from less than 100 characters to over 2,500,000. Assuming an average document length of two kilobytes and a maximum possible amount of text per processing node of two gigabytes, at most 24 megabytes per processing node will be required to store this document information.

In the current implementation, document information structures reside in primary memory. This is justified by the performance gain when document start points are scanned or accessed, which happens very frequently. However, performance would not deteriorate significantly and much memory would be saved if start points were separated out and the rest of the document information structures were stored on disk.

2.4 Searching Algorithm

When searching for a literal term using the super dictionary, the following algorithm is employed. The PADRE user interface broadcasts the search term to all processing nodes, which each perform a binary search of the super dictionary words using a case insensitive form of the term. If a matching word is found in the super dictionary, the corresponding record within the word frequency and index offset file for the super dictionary is read from disk. This record specifies for each sub-collection whether or not there were any matches, and if so, at what offset in the corresponding index file the match data may be found.

For each sub-collection with matches, the relevant pointers are read directly from the index file into memory. If the search term indicates no special case requirements and specifies no terminating punctuation, the selected index pointers are all included in the resulting match set. Otherwise the top six bits of the match are examined, and only those which match the required



```
For efficiency, PADRE keeps the first part of the
super dictionary in primary memory.  Its size is d
ependent upon the combined vocabulary.   At presen
t a fixed amount of space (24 characters) is alloc
ated for each distinct word, regardless of length,
 trading off wasted space for improved searching s
peed.  The results of experiments conducted to est
imate the rate of growth in Vocabulary with growth
 in collection size are reported later.  Once the
super dictionary has been created, component dicti
onaries may be discarded. ........
```

Figure 2: Index, dictionary and superdictionary structures associated with a single processing node. The raw text of sub-collection 2 is represented at the bottom. It has been indexed and a dictionary has been prepared. The information from the latter has been merged into the super dictionary shown at the top. The dictionary may now be discarded. In this example, “ventriloquist” which occurs in at least one of the sub-collections but not in sub-collection 2, has no entry in the dictionary for sub-collection 2 but has a zero entry in the column of the superdictionary corresponding to sub-collection 2. The two small fields in each index entry record the capitalisation of the indicated term occurrence (all lower case in both examples) and the class of character which terminates the word.

specifications are included. If the case or punctuation required is not one of the standard seven encodings, it becomes necessary to examine the raw data using the bottom 26 bits of the pointer as an offset into the corresponding text file.

3 Experiments With The Current Prototype

3.1 Current Status Of The Implementation

At the time of writing, a utility (PARSON) to build indexes, dictionaries and superdictionaries on secondary storage has been developed and is fully functional. All PADRE functions have been extended to permit operation over the new structures except for regular expressions and for strings not constrained to start on a word boundary. The extended modes are capable of processing the majority of queries in ANU's 1995 TREC-4 entry and produce similar (but not identical) results to the original PADRE modes. The differences are due to differences in handling search terms containing punctuation such as 'U.S.A.'

3.2 I/O Costs In Dictionary Building

As noted in table 1, the ANU AP1000 has 128 processing nodes but only 32 of them have physically connected disks. We wished to explore how much time could potentially be saved in index and dictionary building if faster disks were available or if each processing node had its own disk. Figure 3 shows how index and dictionary building and storing time changes as the number of nodes changes, over the 162 Mbyte Ziff collection from CD2 of the TREC data.

Figures 3 and 4 show that the time taken to compute the index and dictionary for a collection dwarfs the I/O costs. The percentage of the time spent in computation rises slowly from about 5% to about 12% as the number of processing nodes increases from 32 nodes (one disk per node) to 128 nodes (one disk per four nodes). If all nodes were equipped with a disk, we would expect that the I/O cost would remain at 5% of the overall time, as both I/O and computation should scale linearly as the number of nodes was increased. The amount of text decreases from roughly 4 Mbytes per node with 32 nodes down to roughly 1 Mbyte per node with 128. The elapsed time taken per Mbyte of text appears to remain roughly constant at 70 seconds for one processing node. This time has increased significantly from the 23 Mbyte/sec per node reported in [6] due to case insensitivity and to recording of capitalisation pattern and terminating punctuation information.

3.3 Scalability Of Index And Dictionary Building

Figure 5 demonstrates how increasing the quantity of text changes the time taken to construct indexes and dictionaries for a collection with a fixed number of processing nodes (128). This was done by breaking the WSJ collection from TREC CD2 into separate components (for the years 1990, 1991, and 1992) and indexing them separately, and then in combination to form the complete WSJ collection.

Here again, each node takes a roughly constant time of approximately 68 seconds per megabyte to build and store index/dictionary information.

We are currently unable to present more information on the extent to which disk-based PADRE operations scale with the number of disks available. PADRE structures on disk are stored in parallel HiDIOS [12] files striped across all disks. Consequently experiments using subsets of the available disks would be difficult to conduct and are likely to be unpopular with other users of the system.

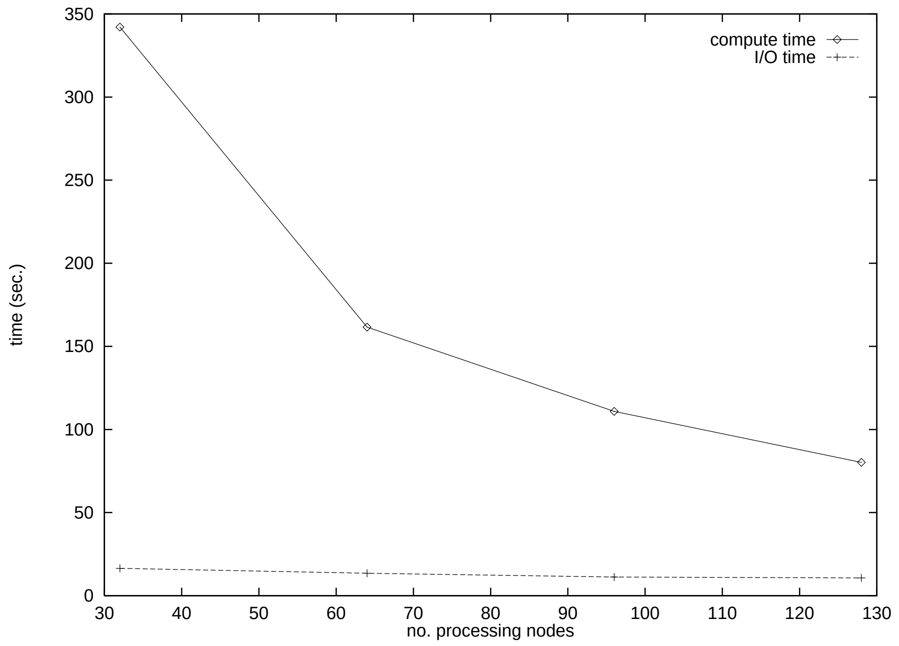


Figure 3: Index/dictionary building and storing costs for the Ziff collection from CD2 of TREC (162 Mbytes after removal of manual indexing terms) plotted against increasing numbers of nodes (32, 64, 96, and 128). The I/O operations in index and dictionary building occur in blocks rather than being interleaved with the computation. This allowed I/O costs to be measured as elapsed times on the front-end. Figures reported thus represent the maximum time for all of the nodes and are likely to slightly over-estimate the time taken. (AP1000, 32 disks)

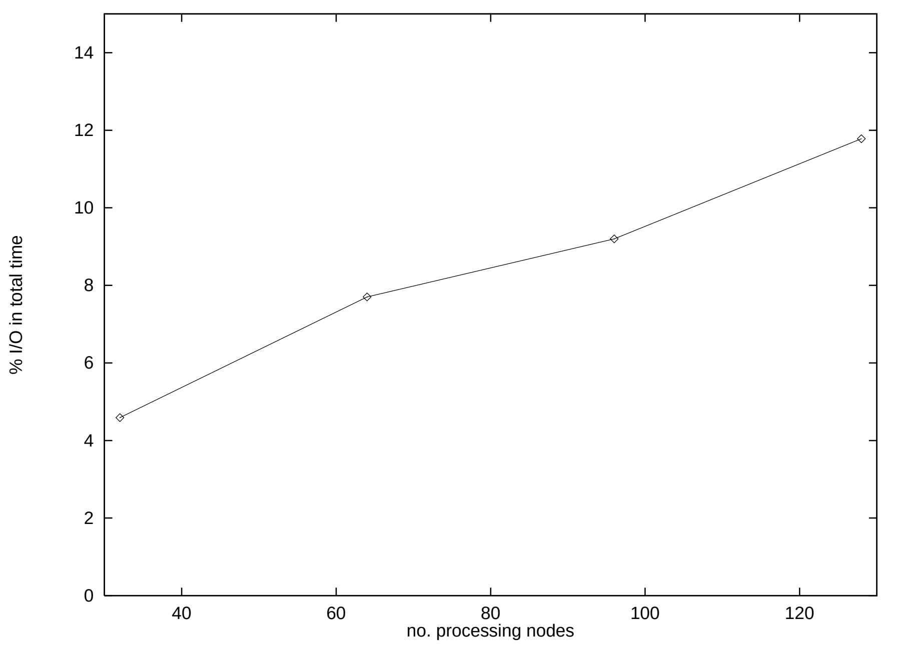


Figure 4: The same data as in figure 3 showing I/O time as a percentage of total time.

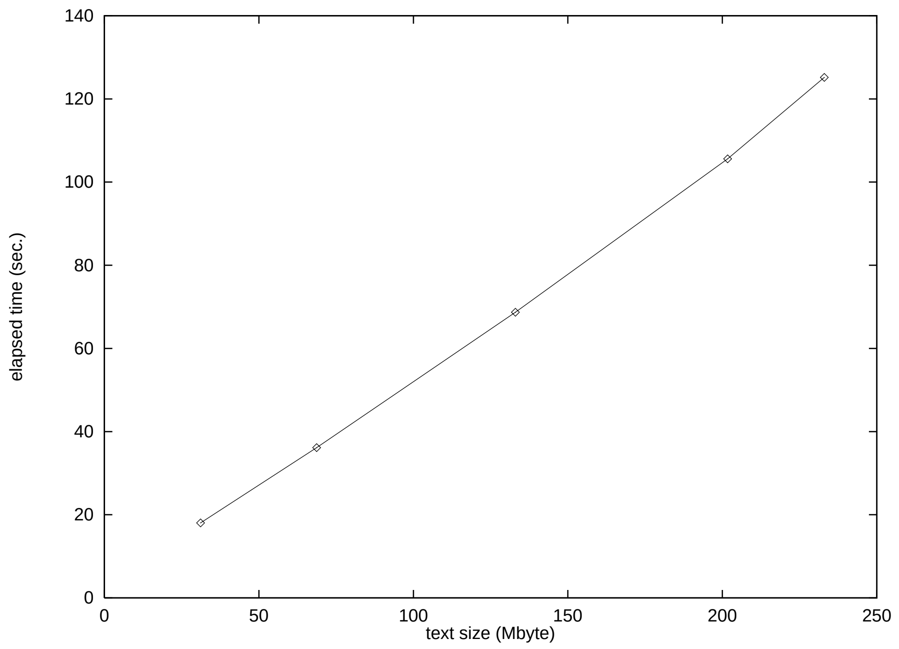


Figure 5: Combined index and dictionary building and storing costs plotted against increasing quantities of text in the WSJ collection from CD2 of the TREC data. (AP1000, 128 nodes, 32 disks.)

3.4 Cost Of Building Super Dictionaries

As can be seen in Figure 6, the first step in creating a super dictionary, essentially copying the dictionary information for the first sub-collection with some additional empty entries, is much cheaper than the subsequent steps. The latter involve merging two files into one and creating new records for each previously un-encountered word in the dictionary being added. Interestingly, when the first point is excluded the remaining five points are well approximated by a straight line with a slope of 7.53 Mbytes/sec. or 136 seconds per gigabyte.

3.5 Performance Of Super Dictionary Searching

Tables 2 and 3 compare the speed of basic term location in the Full Text Scanning (FTS) and Super Dictionary (SD) methods. Terms were chosen to give a range of lengths and a very wide range of collection frequencies. Table 2 reports speeds for case-insensitive searches. Table 3 reports speed of case-sensitive searches for capitalised forms of the same terms. Elapsed times are reported in seconds, first for the full text scanning method and then for superdictionary method.

The speed of the FTS version is primarily dependent on the length of the term, as would be expected from a Boyer-Moore-Gosper derived algorithm. In contrast, super dictionary search time is heavily dependent upon the number of occurrences of the term. The binary search time for the dictionary is neglible (as shown by the results for a word which does not occur in the textbase). The word `Fantasia` occurs roughly once per processing node, necessitating a super dictionary file seek and read operation, and a single collection seek and read operation. For more frequently occurring words such as `the`, there are likely to be matches in each processing node on each sub-collection, necessitating multiple sub-collection seek and read operations. The more collections involved, and the more reading that must be performed from each collection

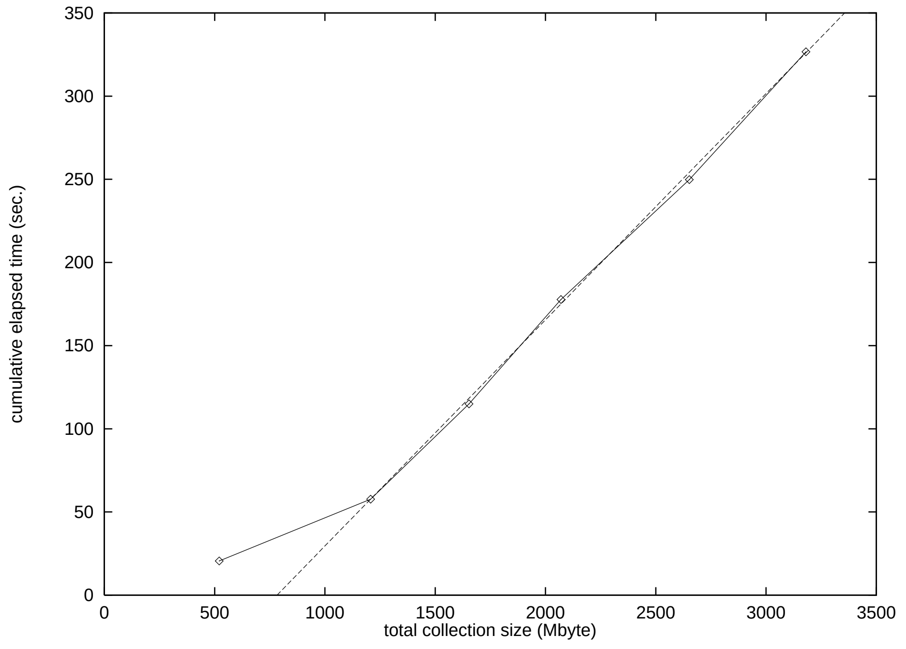


Figure 6: Cumulative time taken to build a superdictionary from component dictionaries plotted against resulting total collection size. Each CD of the TREC collection was divided into two pieces each of approximately 0.5 gigabytes. These pieces were separately indexed and the results combined into a six-way super dictionary for the entire TREC collection (3 CDs). The dashed line shows a linear approximation to the data points excluding the first. (AP1000, 128 nodes, 32 disks)

term (frequency)	`fantasma` (0)	`fantasia` (184)	`god` (10,136)	`america` (83,826)	`the` (16,657,415)
time (FTS)	4.49	4.58	7.16	4.83	7.87
time (SD)	0.01	0.19	0.84	1.09	3.95

Table 2: Comparison of case-insensitive searching performance of full text scanning and superdictionary methods. Elapsed times in seconds as measured on the front end. (AP1000, 128 nodes, 32 disks, CD2 and CD3 of the TREC collection.)

term (frequency)	`Fantasma` (0)	`Fantasia` (145)	`God` (9,420)	`America` (53,829)	`The` (2,331,953)
time (FTS)	4.31	4.41	7.03	4.73	7.31
time (SD)	0.01	0.20	0.89	1.03	4.00

Table 3: Comparison of case-sensitive searching performance of full text scanning and superdictionary modes of PADRE. Elapsed times in seconds as measured on the front end. (AP1000,128 nodes, 32 disks,CD2 and CD3 of the TREC collection.)

```
{proximity 1000}
{wsmode start}
{casesensitive 0}
{weight 0}
anyof "economic |economical |profit |profitable |profits |dollars "
anyof "recycle |recycling |recycles |reprocess |reprocesses |
reprocessing |conversion |converting |converts "
anyof "glass |paper |plastic |aluminum |cardboard "
{weight 5}
near 3
top 20
```

Figure 7: Query 1: The basic test query. Each `anyof` produces a set of matchpoints which becomes an operand to the `near 3` operator.

Method	Elapsed time (sec.)
Full text scanning	36.59
Using 8-way super dictionary	10.64
Using 4-way super dictionary	7.68

Table 4: Processing the test query over 2 gigabytes of data (CD2 and CD3 of the TREC collection) using full text scanning and super dictionary modes. Each time is the average of five observations. (AP1000, 128 nodes, 32 disks.)

chunk's index, the longer the overall time taken.

Case-sensitivity does not have much impact on search times in either mode. Case sensitive super dictionary searches would be expected to slow considerably for search terms containing an odd mix of capitalisation (eg.'UsA') which required reference to the raw text for each possible match.

In all cases, the super dictionary model performs better for these single term searches.

Table 4 compares performance of the two PADRE modes in processing a representative test query, shown in figure 7. This query required the location of 3 groups of alternative literal patterns anchored at word starts (index points), the computation of a 3-way proximity relation (`near`) between the groups, the assessment of relevance of all documents based on matches resulting from these operations and the creation of a ranked list of the 20 most relevant documents. There are a total of 20 literal terms.

In the full text scanning case, the query was run separately over CD2 and CD3 and then the search times were added. Loading times for text data and super dictionaries is not included in the measurements.

Processing the test query over CD1, CD2 and CD3 using the 6-way super dictionary whose construction is graphed in figure 6 took 10.28 sec., again averaged over five runs.

3.6 Relationship Of Super Dictionary To Textbase Size

The size of the super dictionary for a terabyte collection is important in determining whether such a collection can in fact be processed on the target hardware configuration. The super dictionary size depends upon vocabulary of the entire collection as well as upon design decisions such as the number of bytes per entry, the representation of empty fields and the number of sub-collections. It is consequently important for us to know whether there is a fixed upper bound on vocabulary which is assymptotically approached or whether the vocabulary grows essentially without limit as a result of acronyms, codes, mis-spellings and foreign language text.

Zobel et al. [13] present some interesting results on the rate of growth in vocabulary with increasing collection size over the first two CDs of the TREC collection. Naturally, while scanning the collection, the rate of occurrence of previously unseen words is initially high and then falls off. However, Zobel et al. found that even after nearly all of the two gigabytes had been scanned, novel words continued to appear at a rate of about one new word per thousand scanned.

In estimating the size of the superdictionary for a terabyte collection, it is not necessary to know or estimate the novel word occurrence rate at the terabyte level, *provided that the two gigabytes of TREC data constitute a representative sample of likely terabyte collections.* This is because the terabyte super dictionary will in fact be distributed over 512 processing nodes. Each of these nodes will effectively be managing a two gigabyte collection.

For the two gigabyte collection comprising TREC CD2 and CD3, we have observed that the lexicon contains close to 700,000 entries. Zobel et al.'s graph (their figure 2) shows a similar value for the CD1/CD2 combination.

Currently, each record in a 64-way superdictionary for a terabyte collection requires 536 bytes, 24 bytes for the word and 64 offset/length pairs, each represented as 4-byte quantitities. Consequently, the storage estimate for the super dictionary on each node in bytes is 536 times the estimated two gigabyte vocabulary size or 356 megabytes, giving a total superdictionary storage requirement of 178 gigabytes or about 17% of the size of the raw text.

As an aside, among the CD2/3 vocabulary, around 40% of words occur only once; 55% no more than twice; and almost 70% occur less than 5 times in the total of roughly 300 million words. The vast majority of words occurring only once appear to be misspellings and other typographical errors (such as failing to insert spaces between words) and corruptions due to the data capture mechanisms used. It is virtually impossible to put an upper bound on the rate at which these errors will occur in any given body of text.

4 Discussion

4.1 Will an AP1000+ Really Handle a Terabyte Collection?

Our design allows for a terabyte of data but achieving this goal would require an unrealistic (though possible) near-perfect division of the collection into sub-collections and across processing nodes. Assuming perfect division, the major difficulty in accommodating a terabyte collection on the target machine is likely to be lack of disk space.

By definition, one terabyte of disk is required for the hypothetical raw collection. With the present "index all words" strategy and no compression, indexes typically occupy about two thirds of the raw text size. (For CDs 1, 2 and 3 of the TREC collection, the figure is 67.3%). The size of the superdictionary for a terabyte collection is estimated at 178 gigabytes (see section 3.6) or about 17% of the raw text size. Document attributes as currently defined typically require 3% or less of the raw text size. No other disk structures are needed for query processing, consequently 1.87 terabytes is the estimated total requirement.

At present, the largest disk which can be connected to the AP1000+ nodes has a capacity of about 4 gigabytes, giving a total configuration of two terabytes, just large enough for the needs outlined above. Any additional temporary space required during super dictionary building may be gained by temporarily removing part of the raw text and restoring it later on.

Clearly, there is considerable scope to reduce space needs through the use of more compact representations for the various data structures.

4.2 Performance Predictions

Having established to a fair degree of confidence that a maximal Fujtisu AP1000+ possesses the necessary hardware capabilities for storing and operating on a terabyte of text, we now attempt to estimate the time required to build indexes and dictionaries and the time to search a collection of this size.

We assume, based on some preliminary benchmarking of PADRE on an AP1000+, that the CPU performance of AP1000+ nodes will be a factor of 4 better than that of the nodes we have been using in the experiments described above. The maximal AP1000+ has four times as many nodes as the ANU AP1000, therefore total CPU performance increases by a factor of 16.

As for I/O performance, a maximal AP1000+ has 512 disks compared with 32 on the ANU AP1000, representing a potential improvement of a factor of 16. However, the small disks on the ANU AP1000 are quite slow with a maximum practical transfer rate of only 2.0 megabytes/sec. We expect the larger disks to be twice as fast[3], giving an overall I/O throughput improvement of a factor of 32. The AP1000 installed at ANU achieves approximately 50 Mbyte/sec aggregate bandwidth over 32 disks, so the expected achievable bandwidth over 512 disks on an AP1000+ would be $(512/32) \times 2 \times 50 = 1600$ Mbyte/sec. Further study is needed to ascertain the effect of latencies in the I/O system.

4.2.1 Index and super dictionary construction

The figures reported above (section 3.2) indicate that index/dictionary construction and storing time for the TREC data collections were approximately 70 seconds per Mbyte, with only around 5% of the time due to I/O, implying 67 seconds of CPU time. On the AP1000+, we can expect that this cost will be reduced by a factor of approximately 4, to about 17 seconds per Mbyte. Because of the requirement for near perfect collection division, each collection chunk would occupy approximately 32 Mbyte; thus the estimate for the CPU time to construct an index would be $17 \times 32 = 544$ seconds[4].

The space required by indexes and dictionaries combined is dependent upon vocabulary size but is generally less than the size of the raw text. Let us use the raw text size in calculating how long it will take to perform the I/O necessary to write index and dictionary to disk.

Assuming 512 nodes and disks, the I/O time requrired to store the index and dictionary for a sub-collection would thus be $(512 \times 32)/1600 = 10$ seconds.[5] Overall then, for each sub-collection, the time take to produce the necessary files for constructing a super dictionary would be 554 seconds. Since there would be 64 collections, the total time for producing all these files would be $64 \times 554 = 35,460$ seconds or very nearly 10 hours.

[3]Table 1 shows a much higher ratio of peak speeds but this ratio is unlikely to be achieved in our application.

[4]Using a linear approximation, which is likely to involve a small under-estimate. See figure 6 in [6].

[5]this represents a lower percentage of total cost than the 5% quoted above due to the fact that overall I/O speed is expected to increase by a factor of 32 over the test configuration whereas CPU speed increases only by a factor of 16.

Using the linear approximation derived from figure 6, building the super dictionary for a terabyte collection would take $(136 \times 1024 - 106)/3600 = 38.66$ hr. on the ANU configuration, assuming it were possible to do so. The maximal AP1000+ would be a factor of t faster, where $16 \leq t \leq 32$. The factor probably lies closer to 32 than to 16 due to the heavy I/O load in super dictionary building, however, assuming the worst, super dictionary construction may take 2.4 hours.

All up, the cost for constructing a super dictionary and the index/dictionary information for the collections would be in the region of 12.4 hours, though the actual time could vary considerably from this if various assumptions of linearity were not confirmed. Data collections and machine configurations used in our experiments are unfortunately still too small relative to the terabyte case for us to be completely confident in extrapolations.

One important point to note is that if one of the sub-collections were replaced with another different one of the same size, the time required for the system to respond would only be about 1/64th of the overall time or approximately 12 minutes.

4.2.2 Search Performance

Search performance over a terabyte collection is also difficult to predict with accuracy. Search times will continue to be highly dependent on the number of possible matches for any particular term. A binary search over a memory-resident table of about 1.5 million words on each node will be required to locate the relevant row of the super dictionary. The cost of this is negligible.

Thus, to find out that a word does not occur (unlikely though that may be in a terabyte of text), should take no longer than it does currently. Our prediction is that the costs will remain much as reported above for small numbers of matches (up to a few thousand). After that, the time will be heavily dependent on the number of matches found, and the number of different sub-collections involved on each node. The word `the` would be found in each of the 64 collections, and on every node. If total I/O performance increases by a factor of 32 as expected and the number of occurrences of `the` in a terabyte collection is 512 times larger than in the 2 gigabyte sample, then the total time to find all occurrences will increase by a factor of $512/32 = 16$. Since the time currently is around 4 seconds, this would mean that it should take approximately 64 seconds to find all occurrences of `the` in a terabyte textbase!

5 Conclusions and Future Work

With this work we have shown that the PADRE parallel text retrieval architecture may be extended to handle textbases up to 32 times the size of primary memory with relatively small sacrifice in flexibility or functionality for many real applications. The evidence we have gathered suggests that the terabyte limit would be possible, though difficult, to achieve. We are confident that processing of collections of hundreds of gigabytes of text on such a system would be quite practical. We expect that some of the missing functionality, such as anchored regular expressions and literals not constrained to start on aword boundary, will eventually be implemented. However, we recognise that there may still be applications, characterised either by rapidly changing collections or by very complex search patterns, where the full text scanning model will still be preferred. We are currently investigating ways of combining the large data-handling capacity of the super dictionary method with the powerful pattern matching capabilities of full text scanning.

At present we have only a working prototype of the super dictionary system. We are confident that its performance can be improved by a significant factor with relatively little effort at optimisation.

Even the prototype implementation of the super dictionary model has been shown to be acceptably efficient for many applications, both in building structures and in query processing.

Work on larger collections and larger machine configurations is needed to give further validity to our projections and to investigate how well the near-linear scalability observed with the full text scanning is preserved in the super dictionary method.

We are also interested in looking into alternative strategies for managing the derived data with a view to improving efficiency and reducing space. Obviously, the investigation of compression techniques is a high priority. Zobel et al [13] have shown that compression can reduce total disk space required for query processing (not allowing proximity operators) to 40% of the initial raw text size. This ratio is a factor of 4.5 better than the uncompressed approach described here. If similar gains could be achieved in our context, one might hope that queries over a terabyte could be processed on an AP1000+ one quarter of the scale of the maximal configuration. However, in the super dictionary design outlined above, the size of the largest collection which can be handled is limited by the product of the number of nodes and the size of the largest unsigned integer. At the very least, compression will achieve a dramatic saving in disk space.

We conclude that efficiently processing complex retrieval queries over a terabyte text collection is possible on equipment which can be purchased today. We note that the PADRE design is not inherently specific to the machine on which it has been implemented but is believed capable of transplant to any distributed memory multicomputer or even to a network of workstations, though the latter might entail adverse implications for quality of service.

Acknowledgements

The authors are indebted to Dr Paul Thistlewaite for his contributions to the design of the PADRE system, to Professor Robin Stanton for continued encouragement, to Fujitsu Laboratories for assistance with access to equipment and to the National Institute of Standards and Technology (USA) for access to the TREC collection.

The work reported has been supported by the Co-operative Research Centre for Advanced Computational Systems (ACSys) and the ANU-Fujitsu CAP Project.

References

[1] T.C. Bell, A. Moffat, I. H. Witten, and J. Zobel. The MG retrieval system: Compressing for space and speed. *Commun. ACM*, 38(4):41–42, 1995.

[2] D. K. Harman, editor. *Proceedings of the Third Text Retrieval Conference (TREC-3)*, Gaithersburg, MD, November 1994. U.S. National Institute of Standards and Technology. NIST special publication 500-225.

[3] David Hawking. High speed search of large text bases on the fujitsu cellular array processor. In *Proceedings of the Fourth Australian Supercomputing Conference*, pages 83–90, Gold Coast, Qld, Australia, December 1991.

[4] David Hawking. PADDY's Progress (Further experiments in free-text retrieval on the ap1000). In *Proceedings of the First Annual Users' Meeting of Fujitsu Parallel Computing Research Facilities*, Kawasaki, Japan, November 1992. paper ANU-8.

[5] David Hawking. PADRE — a parallel document retrieval engine. In *Proceedings of the Third Fujitsu Parallel Computing Workshop*, Kawasaki, Japan, November 1994. paper P2-C.

[6] David Hawking. The design and implementation of a parallel document retrieval engine. Technical Report TR-CS-95-08, Department of Computer Science, Australian National University, `http://cs.anu.edu.au/techreports/1995/index.html`, 1995.

[7] David Hawking and Peter Bailey. *Parallel Document Retrieval (PADRE) web page.* `http://cap.anu.edu.au/cap/projects/text_retrieval/`.

[8] David Hawking and Paul Thistlewaite. Searching for meaning with the help of a PADRE. In Harman [2], pages 257–267. NIST special publication 500-225.

[9] T. Horie, H. Ishihata, T. Shimizu, and M. Ikesaka. AP1000 architecture and performance of LU decomposition. In *Proceedings of the 1991 International Conference On Parallel Processing*, pages 634–635, August 1991.

[10] Craig Stanfill and Robert Thau. Information retrieval on the connection machine: 1 to 8192 gigabytes. Technical Report DR90-3, Thinking Machines Corporation, Cambridge, Mass., 1990.

[11] Craig Stanfill, Robert Thau, and David Waltz. A parallel indexed algorithm for information retrieval. In *Proceedings of the 1989 SIGIR Conf., Cambridge MA*, page ?, New York, 1989. ACM.

[12] Andrew Tridgell and David Walsh. The HiDIOS filesystem. In *Proceedings of the Fourth International Parallel Computing Workshop*, pages 53–63, London, September 1995. Imperial College/Fujitsu.

[13] J. Zobel, A. Moffat, R. Wilkinson, and R. Sacks-Davis. Efficient retrieval of partial documents. *Information Processing and Management*, 31(3):361–377, 1995.