Scalable Text Retrieval for Large Digital Libraries

David Hawking
dave@cs.anu.edu.au *

Co-operative Research Centre For Advanced Computational Systems
Department Of Computer Science
Australian National University

Abstract. It is argued that digital libraries of the future will contain terabyte-scale collections of digital text and that full-text searching techniques will be required to operate over collections of this magnitude. Algorithms expected to be capable of scaling to these data sizes using clusters of modern workstations are described. First, basic indexing and retrieval algorithms operating at performance levels comparable to other leading systems over gigabytes of text on a single workstation are presented. Next, simple mechanisms for extending query processing capacity to much greater collection sizes are presented, to tens of gigabytes for single workstations and to terabytes for clusters of such workstations. Query-processing efficiency on a single workstation is shown to deteriorate dramatically when data size is increased above a certain multiple of physical memory size. By contrast, the number of clustered workstations necessary to maintain a constant level of service increases linearly with increasing data size. Experiments using clusters of up to 16 workstations are reported. A non-replicated 20 gigabyte collection was indexed in just over 5 hours using a ten workstation cluster and scalability results are presented for query processing over replicated collections of up to 102 gigabytes.

1 Introduction

Whatever the shape of future digital libraries, it is likely that many institutional collections of electronic text will reach the terabyte (10^{12} character) level within the next decade. That such collections are likely to be part of even larger multi-media collections does not at all diminish the importance of free-text techniques for retrieving relevant "documents". In most applications, text captions, descriptions and links still provide the most effective means of retrieving images, movies and sound recordings. Furthermore, text retrieval significantly enhances the catalogue-based searching capabilities of present print libraries.

* The author wishes to acknowledge that this work was carried out within the Co-operative Research Centre For Advanced Computational Systems established under the Australian Government's Cooperative Research Centre's Program.

Techniques for locating relevant information within gigabyte-scale text collections are well-known and quite effective, as witnessed by the results obtained in the annual Text Retrieval Conference (TREC) [6]. Internet search engines perform rapid searches over tens of gigabytes of data but are much more highly regarded for their speed of query processing than for their effectiveness (precision-recall performance). Many libraries presently contain an order of magnitude more text information than that indexed by the largest current internet server.

Designing a retrieval system to operate over electronic text collections of such magnitude is a challenge. Indeed, the author is unaware of any such system described in the literature. Perhaps the closest approaches are those of Stanfill and collaborators [17] and the present author [9]. Both approaches were implemented on expensive supercomputer hardware and neither system was actually tested on collections larger than a few gigabytes. Cahoon and McKinley[5] developed a simulation model for distributed query processing environments and used it to predict behaviour over collections of up to 128 gigabytes. However, their model was only validated on small collections and small numbers of workstations.

Commodity workstations such as Suns, DECs, SGIs, Macintoshes or Intel PCs are much more attractive platforms for text retrieval over gigabyte scale collections because of their low price-performance ratio. Clusters of such workstations, linked by an interconnection network of some sort, also represent a scalable hardware resource whose power can potentially be increased by two or three orders of magnitude, *in unit increments*.

This paper is concerned not only with the design of a text retrieval system capable of eventual operation at the terabyte level but also with the means by which retrieval systems in a digital library may be expanded to cope with three orders of magnitude growth in collection size, from the gigabyte to the terabyte level. Furthermore, a successful retrieval system must provide a reasonably low-cost means of responding to changes in collection content and should be able to present different collection views to different classes of researcher.

The retrieval algorithm presented here is designed to be competitive with other leading systems at the gigabyte level on a single workstation and is also designed to scale to the terabyte level on a cluster of workstations. Using this algorithm, the relationship between collection size and text retrieval performance on both single workstations and clusters of such workstations is investigated.

1.1 Aims

The specific aims of the present work are to:

1. Design and implement a retrieval system which is capable of scalable operation on a large cluster of workstations, capable of reasonably quick response to collection changes and capable of supporting different user views of the collection.
2. Verify that speed of file inversion and of query processing *on a single workstation* and retrieval effectiveness are competitive with other leading systems. Without such validation, claims of scalability will not be particularly interesting.

3. Investigate the relationship between the query-processing speed of the system on a single workstation and the size of the collection.
4. Determine whether the number of workstations necessary to maintain a specified service level scales linearly with increasing data size.

Scalability with growth in query processing load is not addressed here, nor is the relationship between data size and effectiveness.

2 Method

2.1 Overall Design

All experiments used the PADRE97 retrieval system described below. Appendix B of the TREC-4 proceedings [6] was used as the basis for comparing PADRE97 single-workstation performance against that of other leading systems. The relationship between collection size and query processing speed on a single workstation was investigated using artificial collections ranging from one to 20 gigabytes while multi-workstation scalability was investigated using clusters of up to 16 workstations. Query processing times were measured in a reproducible "warm" state, unless otherwise indicated.

2.2 PADRE97 Retrieval System

The PADRE97 retrieval system, designed by the present author, inherits many of the concepts of PADRE [8] for the Fujitsu AP1000 parallel machine but has been redesigned for use on single workstations and clusters.

Index-Building Algorithm PADRE97 generates and uses an inverted file compressed using the Golomb and/or Elias methods recommended by Moffat and various collaborators [2,12]. As pointed out by these authors, compression costs are repaid by reductions in disk I/O. In contrast to their work, here the exact position of every word occurrence (a word is defined as a maximal sequence of letters or digits starting with a letter) is recorded in the form of a `(document-number, offset)` pair. Each entry in the dictionary of all words occurring in the indexed collection points to the startpoint in the compressed index of the block of position pairs for that word. In results reported below, the Elias gamma code is used for both document number differences and offsets within documents.

The algorithm used to build data structures, particularly the compressed index, from the input data requires a number of passes through the input. The first pass builds a table of bundles, a table of documents and a hash table of all distinct words. If Elias coding only is used, this pass also computes the length of the compressed index and the start points within the index for each distinct word. Finally, if disk space is not critically low, the first pass builds a tokenised representation of the input, in which each token consists of a hash table address

and the gap between this word occurrence and the one preceding it. If the more space-efficient Golomb compression method is used in coding document numbers, a second pass is needed to derive the index size and word start points.

Subsequent passes through the tokenised input file write successive windows of the compressed index which is mapped into process virtual memory using the Unix `mmap()` call. The number of passes required depends upon the ratio of compressed index size to the available physical memory. Each additional pass adds the cost of an additional sequential scan of the tokenised input, but working with too few passes imposes a far greater penalty in the form of page thrashing. This tradeoff can be explored by varying the value of the `WIN_SZ` parameter.

Indexing may be performed on collections of data containing no more than about two gigabytes of text and no more than one million documents. However much larger quantities of data may be handled on a single workstation by treating a number of independently indexed collections as a super-collection (see below).

Query-Processing Algorithm Queries are processed by forming matchsets of query term occurrences. Each applicable block of entries in the compressed inverted file is read and decompressed to form an ordered list of pointers into the hypothetical uncompressed text collection. Matchsets may be manipulated using set, proximity and other operators and the relevance scores of documents may be updated on the basis of matchset memberships. In all results reported below, documents are scored using the Cornell approximation to the Okapi BM25 weighting function [14,13] or another *tf.idf* formulation of similar computational cost.

In the retrieval system under study, memory-mapping of the inverted files during query processing was not possible for data sizes exceeding about 8 gigabytes due to the 32-bit virtual memory addressing limit. For consistency, a slower version of the software which seeks and reads index blocks into a buffer was used to derive all timing results reported here.

Super-Collection Architecture In PADRE97's super-collection model, the query processing system is given a list, in the form of a simple text file, of the names of the component collections. In fact, it is possible for a single server to maintain multiple, possibly overlapping, super-collections for different clients. Individual collections may also be indexed and re-indexed independently, thus substantially reducing the cost of accommodating to changes in the data.

The algorithm for matchset formation described above is extended to process all component collections in turn while producing a single matchset. Matchset members are of course tagged with a reference to the collection from which they came to enable appropriate document scoring.

In experiments reported below, the number of component collections in a single workstation's super-collection ranged from 1 to 40.

Approach to Parallel Programming The software may be configured to run only on a single workstation (`S` version) or as a parallel, multi-workstation

(MP) version based upon the MPI[18] communications standard. The software architecture of the MP version is shown in figure 1.

Parallelisation of the retrieval task is achieved by distributing documents across all workstations. Each workstation is responsible for a super-collection and the total ensemble of documents may be viewed as a super-super-collection. Most elements of query processing may be performed independently but cooperation is needed to achieve global ranking and to form global frequency statistics, if required by the ranking method.

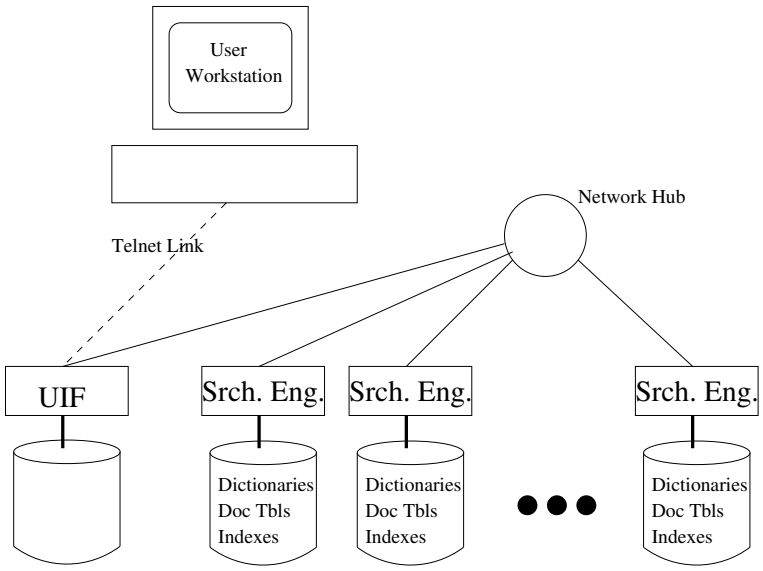

Figure1. The MP version of the software used in multi-workstation experiments provides an ASCII interface shown operating over a telnet link from the user's workstation. The workstation running the user interface (UIF) checks incoming commands and broadcasts them to the search engines. The latter process received commands over their super-collection and, on request, send packets of ranked document identifiers (with scores) to the UIF. The UIF merges the sorted lists and returns the resulting list of document identifiers to the user. Apart from command broadcast and ranking, the only other communication occurring between nodes is global summation of term-document frequencies, if required by the relevance scoring algorithm. If the workstation running the UIF has spare processing power and memory, it may run a search engine as well.

The approach taken to parallel programming was designed to maximise efficiency: communication between workstations occurs only where unavoidable; there are as few synchronisation points as possible; and multiple small messages are buffered into larger single messages. The benefit of these measures is likely

to be felt most on slow interconnection networks, but even where fast networks are used, the minimisation of synchronisation points can dramatically improve performance.

2.3 Hardware Configurations

Single workstation experiments were performed on the systems listed in table 1. Clusters of workstations employed in the study are listed in table 2. The Fujitsu parallel machine runs independent Unix operating systems on each node and is thus equivalent to a workstation cluster with a faster-than-usual interconnection network.

Table1. Single processor configurations used in experiments reported below. In each case disks are directly connected to the processor via SCSI. Disk latency is the sum of average seek time and average rotational delay. Disk bandwidth (BW) is the maximum possible transfer rate from the physical disk (not the disk cache).

Name	Processor	Off-chip Cache	O/S	CPUs	Clock (MHz)	RAM (MB)	Disk lat. (msec.)	Disk BW (MB/sec.)
AP+ node	SPARC 10	-	Linux	1	50	16	14.2	11.3
SPARC 10	SPARC 10	1 MB	Solaris 2	2	50	224	17	6.4
Ultra 1	UltraSPARC	0.5 MB	Solaris 2	1	167	128	17	6.4
Indy	MIPS R4600	-	Irix	1	133	64	13.7	9

Table2. Configuration details for clusters of workstations used in the present study. Reported MPI latencies are half of the round-trip time for a zero-length message sent by the fastest available method. MPI bandwidth is the maximum bisection bandwidth for the given configuration using 200 kbyte messages [15]. One of the ten machines in the SGI lab was a 200 MHz R4400 server, connected to the Ethernet switch by a 100BaseT link. Although it was somewhat faster than the other machines, no attempt was made to take advantage of this by allocating it an unequal amount of data. Since the overall time taken is the time taken by the slowest machine, the presence of a faster machine is unlikely to have affected results.

System	Network Details	Processor Type	No. Nodes	MPI lat.	MPI BW
Fujitsu AP+	2-D torus	AP+ node	16	21.7 μsec.	198.4 MB/sec.
SGI lab	10BaseT/Switch	R4600	10	1.5 msec.	5 MB/sec.

2.4 Test Queries

Experience at the TREC conference [4,1] shows that quite long queries (perhaps created from short queries by automatic term expansion) are necessary to achieve high average-precision on research topics such as, "What dangers are posed by fissionable materials in the states of the former Soviet Union". In such cases, very short queries retrieve large amounts of useless material and fail to retrieve the majority of relevant documents. The most successful TREC queries contain of the order of 50 - 500 query terms. Accordingly, in query-processing experiments reported below, times are given for a query (q254) whose length (51 terms) lies within this range but toward its lower end. This query constitutes a stringent test of the scalability hypothesis as scalability results are expected to improve with increasing query length.

Query optimisations such as those discussed by Smeaton and Kelledy [16], Harman and Candela [7] and Brown [3] can substantially improve performance. Questions relating to their application in a distributed environment are not addressed here but it is acknowledged that query optimisation may reduce the number of terms in a query and also affect the balance between computation and communication.

2.5 Test Collection

The primary test collection consisted of CD2 and CD4 of the TREC collection, a total of 2.01 gigabytes. Data sources included Associated Press, Wall Street Journal, Financial Times, Ziff-Davis Publications, the US Federal Register and the US Congressional Record. Larger scale experiments made use of the 20.14 gigabyte TREC VLC collection [?].

Where necessary to create larger scale collections, collection-derived data structures were replicated. Care was taken to ensure that this was done in a way which would prevent optimisations not obtainable with a real collection of the same size.

TREC data is supplied in the form of bundles of documents compressed using the Unix compress utility and organised into a multi-level directory hierarchy on a CD-ROM. Data from the CD-ROMs was copied onto disk prior to processing but the directory layout and bundle format were retained. Consequently, index-building times reported below include the cost of decompressing the input text into memory.

2.6 Timing Methodology

Query processing times which are of most interest are those which are observed in the course of a stream of query processing (warm state times). Times obtained immediately after start-up of the system are likely to be considerably slower, while those obtained by repeated processing of the same query are likely to be much faster. Interestingly, cold-state times may not be reproducible because of operating system retention of memory pages from previous executions.

In query processing times reported below, a warm-up query was used to engineer a reproducible warm state, unless otherwise indicated. Each such query was immediately preceeded by the warm-up query run over the same collection. This query contained a large variety of terms but none from any of the test queries. The primary goal was to ensure that measured times were not subject to uncontrolled variation due to previous query processing.

3 Results

Results are first presented for single-workstation experiments and then for those using clusters of workstations.

3.1 Single-Workstation Results

Data Structure Building Appendix B of the TREC-4 proceedings [6] was used as a guide to the state of the art in indexing performance over two gigabytes of TREC data. The four fastest index-building CPU-time figures reported in the appendix are summarised in table 3. Differences in system configuration and amount of information in the index and lack of information about timing methodology preclude precise comparisons. However, it seems reasonable to conclude that the data structure building performance of the system under study, summarised in table 4 is close to state-of-the-art on comparable hardware. Rates for the SPARC 10 could be expected to improve if the machine were dedicated to the task and the WIN_SZ parameter increased to a larger fraction of the available physical memory.

Table3. Leading indexing performances reported for AdHoc runs in Appendix B of the TREC-4 proceedings. The collection comprised CD2 and CD3, a total of 1,975 MB. Rates are reported in megabytes of raw text indexed per CPU hour.

Group	Index Size	Machine	CPUs	RAM	Rate
George Mason Uni	248 MB	SPARCCentre 2000	18	2048 MB	658
Cornell Uni	731 MB	SPARC 10/512	2	192 MB	564
RMIT/CITRI	140 MB	SPARC 10/514	4	256 MB	494
Dublin City Uni	253 MB	SPARC 20	1	96 MB	439

Despite use of full positional information, algorithms used are reasonably frugal with disk space: Data structures required for query processing over the 1194.4 MB CD4 collection totalled 481.9 MB or 40.4%. When a stopword list was used and terms were stemmed, disk requirements reduced to 273.5 MB or 22.9%. Temporary space required for the tokenised text (including stopwords) was 698 MB but if disk space were limited, the data structures could be built more slowly with zero temporary space requirement.

Table4. PADRE97 index-building speed for single workstation systems using the algorithm described above. Configuration details for systems employed are given in table 1. Times used in calculating rates include collection decompression and building of document table, bundle table, dictionary, inverted file index and lexicon. Index sizes include only the compressed inverted file.

System	WIN_SZ MB	Collection/ Sz. (MB)	Index Sz. MB	Rate(Elaps.) MB/hr	Rate(CPU) MB/hr
SPARC 10 (224 MB)	80	CD2 (863.4)	359.9	317	584
Ultra 1 (128 MB)	50	CD2 (863.4)	359.9	679	1623
Ultra 1 (128 MB)	50	CD4(1194.4)	463.6	680	1576
Ultra 1 (256 MB)	128	CD4(1194.4)	463.6	1155	2042
Ultra 1 (256 MB), stems/stopwd elim.	128	CD4(1194.4)	251.2	1084	1408

Query Processing Speed Figure 2 plots the deterioration of query processing speed on a single workstation (Ultra 1) as data size is increased. Table 5 compares the times taken to process query `q254` over the two gigabyte test collection on various single workstations.

Table5. Query processing times for query `q254` on single workstation systems (1,000 documents ranked). Configuration details for systems employed are given in table 1. Times reported are elapsed times as would be observed by the user and are averaged over 5 runs.

System Employed	Collection	Coll. Size	Term Proc.	Ranking	Total
AP+ node	CD2/CD4	2.01 gB	151	51	202
SPARC 10	CD2/CD4	2.01 gB	18.6	10.3	26.8
Ultra 1	CD2/CD4	2.01 gB	10.5	5.3	16.8

I/O Load During Query Processing Processing query `q254` over one copy of CD2/CD4 required 95 seeks within the compressed index files. Using the tabulated disk latency for the disk used on Ultra 1, this represents a total of 1.62 sec. The 51 terms in the query result in a matchset with 661,041 members. It is estimated that this would be represented in a total of 1.73 MB of compressed pointers which could be expected to take 0.27 sec. to transfer into memory. Consequently, 1.89 sec. out of the total query processing times reported in table 1 are due to I/O from the index file, about 11% of the total in the case of Ultra 1. Half of the seek times and half the dictionary lookup times, or 5% of the total

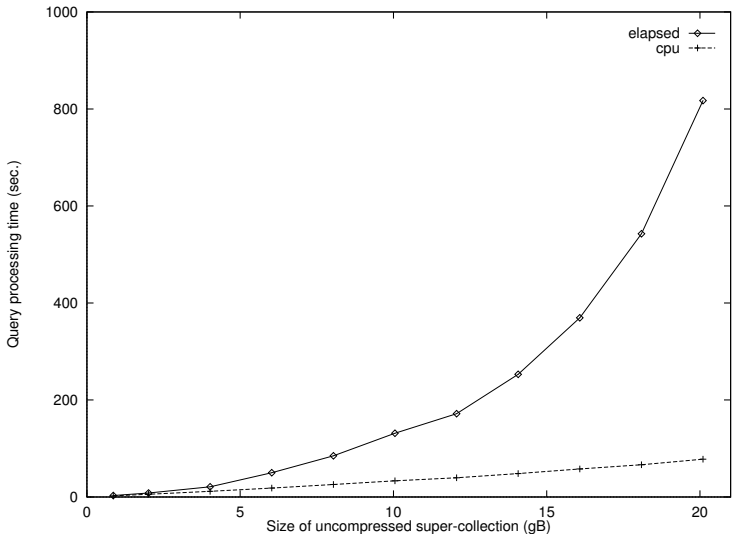

Figure2. Total warm-state query processing time for query `q254` on Ultra 1 for various collection sizes.

query processing time could be saved using a single merged index instead of the two-component super-collection.

Query Processing Speed Compared to Other Systems For purposes of comparison with the query-processing times quoted in Appendix B of the TREC-4 proceedings, it is noted that the elapsed time of 26.8 seconds for the SPARC10 system (table 5) included 21.8 seconds of CPU time. This time is three orders of magnitude faster than the slowest system and would have ranked sixth out of 21 category A adhoc systems listed. Nevertheless, two groups did report average "searching" times of the order of one CPU second. However, one of the latter (CITRI [19]) used very much shorter queries, while the other (ETH [11]) used a totally different method for which data structures required two orders of magnitude longer to build. Assuming that the quoted times in fact represent full query-processing times and that a significant performance gap remains after correcting for length of queries, work is needed on the present software in order to validly claim "state-of-the-art" status.

Query processing speed could be improved by limiting the number of document accumulators. Not only would this reduce the scale of the sort necessary for ranking but it would improve locality of memory referencing and reduce paging. Smeaton and Kelledy [16]) found that this optimisation did not adversely affect retrieval effectiveness. Removing the term-position information from the

inverted file would significantly reduce the CPU cost of decompressing index blocks and also decrease paging overheads.

Scalability of Single Workstation Query Processing Figure 2 shows that while the increase in cpu time required to process a query rises more or less linearly with increasing super-collection size, the elapsed time increases sharply for sizes over about 4 gigabytes. This is clearly due to paging. It seems highly likely that the "linear" section of the elapsed-time plot could be extended by increasing the amount of system RAM. In fact, RAM capacity of Ultra 1 could be increased by a factor of eight which would probably be sufficient to reduce elapsed time for the 20 gigabyte collection down close to the CPU time recorded for that collection size. Application of some or all of the optimisations discussed in section 3.1 together with compression of dictionary and document table would be expected to extend the near-linear portion of the elapsed time graph in figure 2.

In the absence of memory effects, approximate linearity with data size would be expected of the super-collection model over a wide range of collection sizes. Better than linear scaling might be achievable by merging dictionaries and combining index files. However, there would be very considerable loss of flexibility and of responsiveness to collection changes. It may also be necessary to change index design parameters, with possible loss of index-building performance.

Query Processing Effectiveness To verify that the system described is capable of achieving good precision-recall performance, a simple automatic query generator was used to generate PADRE97 queries from the full topic descriptions for TREC-3, TREC-4 and TREC-5. These were processed using the Cornell approximation to the City University scoring model [14,13] and a relevance feedback scheme which was tuned only on the TREC-5 data. As may be seen in table 6, results obtained would have been quite competitive in each case.

Table6. Precision-Recall Performance on TREC Tasks. The best runs from each group submitting a run in the Automatic Adhoc, Category A section of TREC were ranked by average precision. The group rank column shows where in this ranking the present run would have occurred had it been submitted as an official run.

Task	Average Precision	Group Rank	Remarks
TREC-3	0.3537	3	
TREC-4	0.2337	5	Short topics
TREC-5	0.2493	3	

It was also confirmed that PADRE97 version `S` on Ultra 1 and version `MP` on SGI lab produced identical precision-recall results on the TREC-4 task.

3.2 Multiple-Workstation Results

Query Processing Speed Figure 3 shows the scalability of query processing on the Fujitsu AP+ over a collection small enough to avoid large-scale paging. The cost of running the `MP` version as opposed to the `S` is that average query processing time for `q254` after `qwarmup` is 7.67 vs. 6.26 sec, an increase of 22%. In fact, matchset formation is slower (3.27 vs. 2.85 sec.) but ranking is considerably faster (2.99 vs. 4.82 sec.).

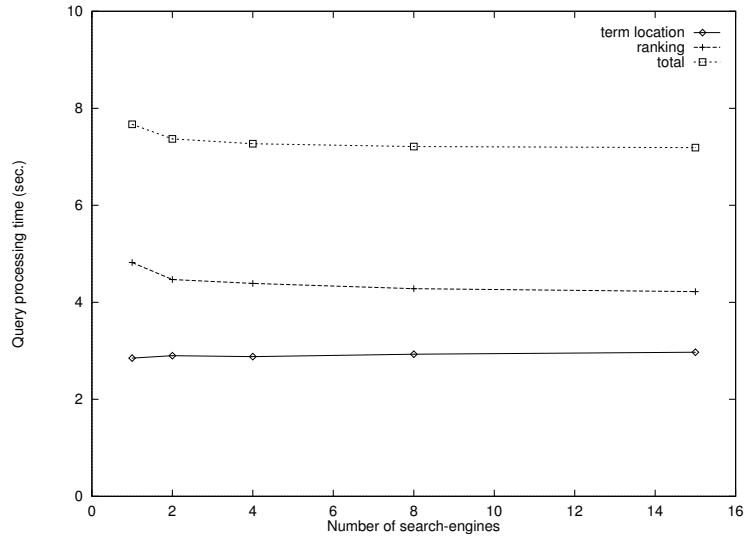

Figure3. Times taken to process query `q254` using n nodes of the AP+, each with a local copy of the 242 MB WSJ sub-collection. Each point is the average of 5 observations.

As may be seen, query processing time in fact reduces slightly as the collection size (and the cluster size) is increased. This is because a slight increase in term location time (due to communication during computation of global frequencies) is compensated for by a larger decrease in ranking times due to a reduction in buffer latencies. If a double-buffering strategy were employed to improve overall ranking times, the reduction in ranking time with increasing cluster size would be expected to disappear.

Let t_S be the time taken by the `S` version of the software to process a standard query over a standard collection on a standard workstation configuration. Let t_{c1} be the time taken in communication between the UIF and the search engine in the case where the `MP` version of the software is run with a single search engine on a standard workstation. The total query processing time for the `MP` version

in this case is thus: $t_{MP1} = t_S + t_{c1}$, and the initial cost of using the `MP` version relative to the `S` is given by $(t_S + t_{c1})/t_S$.

Assuming no load imbalance, the time t_{MPk} to process the query using k search engines over a k-times larger collection is determined by the increase in communication costs ($\Delta C_k = t_{ck} - t_{c1}$). C_k is a function of k which depends upon the characteristics of the network. On certain networks it may rise sharply due to saturation or contention. The ratio $S = t_{MP1}/(t_{MP1} + \Delta C_k)$ may be used as a measure of scalability. Full scalability is indicated by $S = 1.0$. Figure 3 indicates that $S > 1.0$ for the conditions of the experiment but this is due to the buffer strategy described above. A more realistic value is likely to be quite close to unity, however.

Scaling the total times shown in figure 3 by a factor which would cause the version `S` time for an AP+ node over the sub-collection to coincide with that for the `S` version on Ultra 1 allows an estimate to be obtained of query processing times obtainable on a cluster of Ultra 1 workstations in which a separate search engine node is assigned to each 2 gigabytes of collection size. Figure 4 compares the relationship of these hypothetical scaled values to collection size with both the observed processing times on Ultra 1 and with the expected values if the Ultra 1 memory were increased to the extent necessary to avoid increased paging.

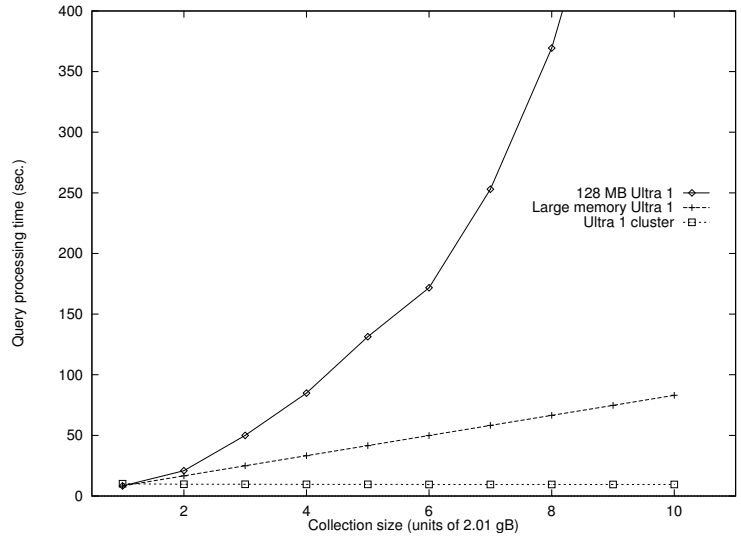

Figure4. Elapsed query processing times for processing collection sizes measured in 2.01 gigabyte units on three different systems: Ultra 1 (observed times); Ultra 1 with memory hypothetically increased in proportion to the data size (projected times); a cluster of Ultra 1s with one search engine for each unit of collection size (scaled up from the data in figure 3).

Table7. Ultra 1 cost relativities between workstations, memory and disks. (Taken from current Sun Microsystems Australia list prices.)

Upgrade	Scaled cost
Add 128MB/4 gB workstation (and fraction of Ethernet hub)	1.00
Increase physical memory to 1 gigabyte	3.15
Add disk and controller	0.18

3.3 Scalability of Data Structure Building

If a collection to be indexed is broken up into sub-collections and each is indexed on an identical independent workstation then it is clear that the time to index the whole collection is the same as the maximum time taken to index one piece. Assuming that indexing speed is proportional to data size, then equal-sized sub-collections will yield the best possible result. Otherwise, overall indexing time will depend upon the load imbalance ratio LI which is the ratio of largest to average data holding. Data structure building is trivially scalable with increasing data size because indexing of one sub-collection is totally independent of the others. Scalability should be optimal when $LI = 1.0$.

4 Larger Scale Experiments

The following experiments used 20.14 gigabytes of real data from the TREC Very Large Collection (VLC) [?] and the SGI lab cluster of workstations.

4.1 Indexing 20 Gigabytes

The 34 collections comprising the VLC were distributed across the disks of the ten machines according to the best-balanced arrangement encountered during one billion random trials. The calculated load imbalance was 1.057. Because of various system and software limits, it was found necessary to divide the larger collections into separately indexed pieces. Each machine commenced indexing at the same time and indexed each of its collections, one after the other. Unfortunately, table overflow problems were experienced on one machine and its indexing was re-run and timed separately.

Results are shown in table 8. Note that equalising data sizes across the nodes was not as effective in balancing load as had been hoped due to differences in indexing difficulty across collections. In particular, the USENET news collections contained a very large number of indexable words.

Table8. Indexing the VLC on a cluster of nine identical workstations and a somewhat faster server. Elapsed (wall clock) times are reported. Indexes were built using word-stemming and excluded stopwords. They included full positional information. Quoted time to index the VLC is the time taken by the slowest workstation and the indexing rate is obtained by dividing the total amount of data by the time taken by the slowest workstation.

Time on server (min.)	Time on fastest w/s (min.)	Time on average w/s (min.)	Time on slowest w/s (min.)	Time to index VLC (min.)	Indexing rate gB/hour
162	203	243	305	305	3.96

4.2 Query Processing Over 102 Gigabytes

In order to be able to claim with reasonable confidence that the query processing system under study is capable of scaling to terabyte level collections, query processing was performed over a collection of only one order of magnitude less than this (including replications).

The data structures built in the previous section were replicated across the SGI lab cluster so that each workstation held all the data structures necessary to process queries over the VLC. Unfortunately, virtual memory and swap space limits prevented processing of queries over more than about 60% of the VLC on a single workstation. A conservative subset of the indexes corresponding to just over 10 gigabytes of text was chosen and used to synthesize larger distributed collections by replication. Each workstation thus processed the queries over a private copy of a 30-component non-replicated super collection totalling 10.2 gigabytes.

Four shorter queries comprising 4 to 16 (average 8.25) word stems were used to test query processing speed over these synthetic collections. The average time to process the four queries over 10.2 gigabytes on a single workstation using the `S` version of PADRE97 was 322 seconds. The corresponding time for the `MP` version processing the queries on the same workstation and same super collection with the user interface running on a second workstation was 314 seconds. There appears to be no significant difference between the two times.

Figure 5 shows how query processing time varies as the number of workstations is increased in step with the total data size. Very good linearity is observed up to nine workstations. The much higher time taken by ten workstations is almost certainly caused by interference between the user interface and retrieval engine processes running on the same machine, aggravated by intense paging. Clearly, a separate workstation should be used to run the UIF. A smaller lower-cost configuration would almost certainly suffice for this purpose.

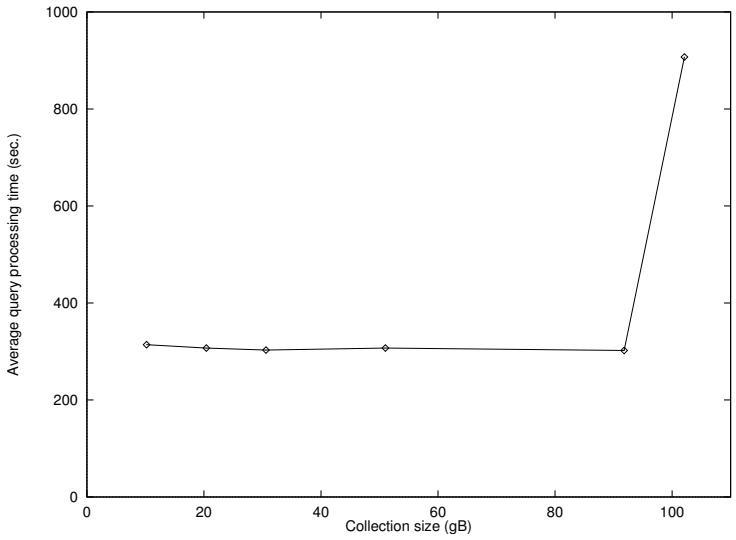

Figure5. Observed times to process four shorter queries over collections of up to 102 gigabytes on clusters of two to ten 64MB SGI Indy workstations connected by Ethernet. Each point shown represents the average of the elapsed processing times for one run of the four queries in sequence. The runs presented were performed when the laboratory was believed to be otherwise idle. For each 10.2 gigabyte increase in overall collection size, an extra workstation was used in searching. In the 102.1 gigabyte case, the same workstation (the server) ran both the user inteface process and a retrieval engine. No warm-up query was used but the huge ratio of collection size to physical memory (approximately 320) means that the effect of warm-up would be slight.

5 Conclusions and Further Work

This study has demonstrated that multi-workstation query processing time remains almost constant provided that the number of workstations engaged in searching is increased in proportion to the size of the overall collection. By contrast, on a single workstation, query processing time increases as a much greater than linear function of data size once overall collection size exceeds physical memory by a factor of more than about 32. The cost of running the `MP` (multi-workstation) rather than the `S` version of the software was found to be small.

Query processing scalability results reported here are optimistic due to the fact that each node used an identical collection. In more realistic use there will be possibly significant imbalance between nodes. If query processing time is proportional to data size, then the workstation with most data will take LI times as long as the average, thus delaying the overall result and wasting resources. This is not an argument against use of parallelism, but rather a warning against careless application of parallelism.

Because parallelism is being used here to cope with an increased problem size rather than to perform a fixed task faster, the diminishing returns inherent in Amdahl's law are avoided. Speeding up the serial parts of the algorithm would increase the overhead of moving from the single-workstation to the multi-workstation version but should not harm the scalability of the algorithm. Scalability will, however, be adversely affected if communication costs increase.

Provided that the circumstances of the present experiment remain applicable, and the same software is used, figure 4, table 7 and the I/O cost analysis in section 3.1 suggest the following:

> If queries are currently being processed over a 2 gigabyte collection on a 128 MB Ultra-1 workstation and data size is to be increased by a factor of ten, then:
> - 9.5 units of cost invested in additional workstations[1] can be used to preserve current response times, provided near-perfect load balance is achieved;
> - 3.15 units of cost invested in upgrading memory on the single workstation will allow query processing times to grow by a factor of ten but probably no more;
> - 0.18 units of cost carefully invested in disk parallelism could be expected to reduce elapsed times by a factor of two at best. In the absence of other upgrades, query processing times would thus still grow by a factor of 50.

In reporting scalability results, there is a danger that the performance of a parallel algorithm running on a single node will be much worse than that of the best serial algorithm. The present work has attempted to avoid this by designing a serial algorithm, verifying that it is competitive in both speed and effectiveness with other systems and measuring the performance difference between this algorithm and a parallel version running on a single search node. Only then was multi-workstation scalability investigated.

In fact, it appears that the current serial algorithm, while competitive with most TREC-4 submissions using similar hardware, may be much slower at processing queries than were the two fastest submissions. More information is needed for the submissions in question to enable determination of the true magnitude of the difference remaining after correction for query length.

The inclusion of term positions in the inverted file structure used here naturally increases data sizes and processing times but allows the use of proximity operations and distance-based relevance scoring [10]. Optimisations listed in section 3.1 which can be applied without harm to effectiveness are likely to be applied in future versions of the software.

Despite the caveats mentioned above, results presented here suggest that parallelism will be essential to enable full-text searching over the terabyte-scale text collections forecast for future digital libraries. Although experiments have to

[1] Assuming that a small machine to run the UIF can be bought for half the cost of a retrieval workstation

date been conducted with no more than 16 workstations in a cluster, it appears that communication costs are low enough on the Fujitsu AP+ to permit scaling to the terabyte level. The demonstrated ability of PADRE97 software to process queries over collections in excess of 100 gigabytes on as few as ten workstations gives further confidence that this goal is achievable with larger but still practical clusters.

Scalability of text retrieval with increasing data size has been seen to depend upon the ratio of increase in communication costs to single workstation processsing speed. This suggests that scalability results presented here will remain valid provided either: that network performance grows in proportion to increasing workstation power, or that data sizes handled by each workstation grow in proportion to increasing workstation power.

It is worth noting that, to the knowledge of the author, the 102 gigabyte synthesised collection used in experiments reported here is larger by an order of magnitude than that used in any similar experiments reported in the literature to date. It is believed to be larger by a factor of approximately two than the largest amount of data indexed by the of the current web search engines. Handling such large volumes of data caused considerable logistical difficulties!

Further work is needed to confirm expected scalability on larger clusters of workstations and even larger collections and to explore how scalability on large clusters is affected by slow interconnection networks.

ACKNOWLEDGEMENTS

Thanks are due to Robin Stanton, Kathy Griffiths, Paul Thistlewaite and Kerry Webb for comments on the manuscript and to the U.S. National Institute of Standards and Technology and various copyright holders for access to the TREC data collection. Thanks also to David Sitsky for assistance with MPI and to Henry Gardner and Ben Evans for access to the SGI laboratory.

References

1. James Allan, Lisa Ballesteros, James P. Callan, W. Bruce Croft, and Zhihong Lu. Recent experiments with INQUERY. In Harman [6], pages 49–63. NIST special publication 500-236.
2. T.C. Bell, A. Moffat, and I. H. Witten. Compressing the digital library. In *Proceedings of the Digital Libraries 94 Conference*, Texas A&M University, College Station TX, 1994. `http://abgen.tamu.edu/DL94/paper/bell.html`.
3. Eric W. Brown. Fast evaluation of structured queries for information retrieval. In Edward A. Fox, Peter Ingwersen, and Raya Fidel, editors, *Proceedings of SIGIR'95*, pages 30–38, Seattle, Washington, July 1995. ACM Press, New York.
4. Chris Buckley, Amit Singhal, Mandar Mitra, and Gerard Salton. New retrieval approaches using SMART: TREC-4. In Harman [6], pages 25–48. NIST special publication 500-236.
5. Brendon Cahoon and Kathryn S. McKinley. Performance evaluation of a distributed architecture for information retrieval. In Hans-Peter Frei, Donna Harman, Peter Schäuble, and Ross Wilkinson, editors, *Proceedings of SIGIR'96*, pages 110–118, Zurich, Switzerland, August 1996. ACM Press, New York.

6. D. K. Harman, editor. *Proceedings of TREC-4*, Gaithersburg MD, November 1995. NIST special publication 500-236.
7. Donna Harman and Gerald Candela. Retrieving records from a gigabyte of text on a minicomputer using statistical ranking. *Journal of the American Society for Information Science*, 41(8):581–589, 1990.
8. David Hawking. The design and implementation of a parallel document retrieval engine. Technical Report TR-CS-95-08, Department of Computer Science, The Australian National University, Canberra, `http://cs.anu.edu.au/techreports/1995/index.html`, 1995.
9. David Hawking. Document retrieval performance on parallel systems. In Hamid R. Arabnia, editor, *Proceedings of the 1996 International Conference On Parallel and Distributed Processing Techniques and Applications*, pages 1354–1365, Sunnyvale, California, August 1996. CSREA, Athens GA.
10. David Hawking and Paul Thistlewaite. Relevance weighting using distance between term occurrences. Technical Report TR-CS-96-08, Department of Computer Science, The Australian National University, `http://cs.anu.edu.au/techreports/1996/index.html`, 1996.
11. Daniel Knaus, Elke Mittendorf, Peter Schäuble, and Páraic Sheridan. Highlighting relevant passages for users of the interactive SPIDER retrieval system. In Harman [6], pages 233–244. NIST special publication 500-236.
12. Alistair Moffat and Justin Zobel. Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems*, 14(4):349–379, 1996.
13. S. E. Robertson, S. Walker, M.M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In D. K. Harman, editor, *Proceedings of TREC-3*, Gaithersburg MD, November 1994. NIST special publication 500-225.
14. Amit Singhal, Gerard Salton, Mandar Mitra, and Chris Buckley. Document length normalization. Technical Report TR95-1529, Department of Computer Science, Cornell University, Ithaca NY, 1995.
15. David Sitsky, Paul Mackerras, Andrew Tridgell, and David Walsh. Implementing MPI under AP/Linux. In *Proceedings of the MPI Developer's Conference, Notre Dame IN*, July 1996.
16. Alan F. Smeaton, Fergus Kelledy, and Ruairi O'Donnell. TREC-4 experiments at Dublin City University: Thresholding posting lists, query expansion with WordNet and POS tagging of Spanish. In Harman [6], pages 373 – 389. NIST special publication 500-236.
17. Craig Stanfill and Robert Thau. Information retrieval on the Connection Machine: 1 to 8192 gigabytes. *Information Processing and Management*, 27(4):285–310, 1991.
18. University of Tennessee (Knoxville). *MPI: A Message-Passing Interface Standard*. `http://www.epm.ornl.gov/$_{\tilde{}}$walker/mpi/index.html`, 1995.
19. Ross Wilkinson, Justin Zobel, and Ron Sacks-Davis. Similarity measures for short queries. In Harman [6], pages 277–285. NIST special publication 500-236.