PADRE for COWs

David Hawking
Co-operative Research Centre for Advanced Computational Systems
Department of Computer Science
Australian National University
dave@cs.anu.edu.au *

February 19, 2007

Abstract

Earlier work with the Parallel Document Retrieval Engine was oriented toward parallel machines such as the AP1000, characterised by many nodes, few disks, small memory per node (by current standards), single-user operation and high communication performance, relative to node computational power.

Present generation parallel machines are much more like clusters of workstations (COWs). There are typically fewer nodes but each is more powerful, runs a multi-user operating system, supports more memory and connects to at least one local disk. In general, COWs are characterised by poorer network performance.

PADRE has been redesigned to operate in the COW environment. Indexing and retrieval algorithms and user-interface have been totally replaced, along with the PADRE model of parallelism. PADRE97 minimises communication and synchronisation in order to improve scalability on high-latency clusters. Results are presented to show that the new design has achieved its objectives.

1 Introduction

Early work on PADRE and its predecessors [?; ?] was targeted at the Fujitsu AP1000, a massively parallel but diskless MIMD architecture in which total memory and processing power were large relative to the scale of available text collections and where the ratio of communications speed to processing speed of a node was high.

This environment encouraged the investigation of applications of otherwise impractical full-text scanning methods in text searching and document retrieval. Inverted file indexes were also introduced 1991 and the algorithms used to construct them were based on the assumption that both data and indexes were memory resident. This resulted in very fast indexing. Indexing rates of 10.6 gigabytes per hour [?] and 79.0 gigabytes per hour [?] were observed for a 0.5 gigabyte collection on 64 nodes and for a 2 gigabyte collection on 512 nodes, respectively. By way of comparison, a rate of 2.6 *megabytes* per hour achieved for disk-resident indexes and data on a multi-processor shared-memory system in 1989 [?] was deservedly regarded as a breakthrough in indexing methods for conventional hardware.

When nodes of the AP1000 were equipped with disks it became desirable to move PADRE indexes and data to the disk system in order to increase the amount of data which could be processed by a given hardware configuration. However, indexes were still limited in size because the basic algorithm assumed memory-residency during building. To increase collection sizes further a *super-dictionary* structure [?] was introduced to allow several indexes to be searched in tandem. On an AP1000 system comprising 128 nodes and 32 disks managed by the HiDIOS parallel filesystem [?], the rate of index building (including super dictionary building) declined to 5.3 gigabytes per hour, partly due to disk I/O but also partly due to changes in index format.

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

The scale of collection which could be processed was increased considerably. (The fully memory resident indexing version was restricted to no more than about 0.9 gigabytes on a 128-node system but the disk-resident version was shown to process 3 gigabyte super collections and was capable of processing many times this size). Unfortunately, query processing was surprisingly slow given the amount of hardware employed. The reason identified by **?**] was that fragmentation of the overall index structures due to the super-dictionary model and distribution over 128 nodes resulted in excessive delays due to disk latency.

A complete re-design and re-implementation of PADRE has been undertaken recently with the following objectives:

1. To update PADRE to operate in a modern cluster-of-workstations (COW)[1]. In the target environment, each workstation will provide a local disk and at least 64 Mbytes of RAM but network latencies may be much higher relative to compute performance.

2. To achieve high indexing speed while retaining the full term location information necessary to support PADRE `span` and other proximity operations.

3. To achieve high query-processing speed and high effectiveness.

4. To preserve or improve the scalability properties of the PADRE algorithms and to preserve the ability to respond quickly to updates of subsets of the collection.

5. To dramatically reduce total disk space required.

?] showed that the resulting new version of PADRE:

1. is capable, when run on a single workstation, of retrieval effectiveness, indexing speed and query processing speed comparable to the two or three best TREC [**?**] submissions,

2. appears to scale well on clusters of up to 15 workstations when the number of workstations used is increased in proportion to the size of the collection.

3. is capable of processing collections which are a substantial fraction of a terabyte in size. Runs were actually conducted over index structures corresponding to collections of up to 102 gigabytes.

The present paper presents highlights of the PADRE97 architecture, reports indexing and query-processing results for a real 20 gigabyte collection and draws conclusions from six years of research into parallel text retrieval.

2 New Indexing Structures and Algorithms

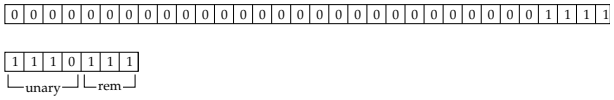

Figure 1: The Elias gamma code. The number 15 is shown represented as a 32-bit integer (top) and in gamma code (bottom). The unary part of the gamma code is a zero-terminated unary representation of $\lfloor \log_2(x) \rfloor + 1$ and the rem part represents $x - 2^{\lfloor \log_2(x) \rfloor}$.

Moffat and various collaborators [**?**; **?**; **?**; **?**] describe fast algorithms for inverting text files which operate in limited memory and require only small amounts of disk space. The algorithms achieve dramatic space savings through the use of word-based Huffman coding of the raw text and Elias and Golomb compression schemes within the inverted file and handsomely repay the cost of the compression through reductions in disk I/O and consequent improvements in elapsed times. Figure 1 illustrates the space savings achieved when coding small integers with the Elias gamma-code.

The new PADRE indexing algorithm and data structures build on the above-referenced ideas and are designed with the following limits in mind:

[1] The Fujitsu AP3000 is here considered to be a cluster of workstations with a high performance interconnection network.

1. No more than 2 gigabytes in any collection[2].

2. No more than 1 million documents in a single collection.

3. No more than 1 million distinct terms in a single collection.

The input data is assumed to be in the same format as that of the TREC CD-ROMs, that is, a directory hierarchy in which each file is a bundle of documents compressed using the Unix `compress` algorithm.

Indexing output includes: Document table, Dictionary and Compressed Index.

2.1 Overview of P97 Indexing

The indexing algorithm is characterised by two broad phases:

1. Construction of a hash table containing entries for each distinct word or stem[3]. This process derives all the information, such as compression parameters and file locations, necessary for the second phase. This phase is characterised by sequential file I/O.

2. Writing the compressed index file. This phase is characterised by highly random writing of the output file and can potentially consume huge amounts of time if input data size is large.

Performance during the index-writing phase is very much dependent upon the relationship of compressed-index size to that of available physical memory. PADRE avoids page thrashing by dividing index-writing into a number of passes. The entire output file is memory-mapped and a number of passes through the input are performed. In each pass, the entire input is re-scanned and a contiguous window of the index file is written. Word occurrences falling outside the writing window are ignored. To avoid the cost of re-stemming and re-hashing during each re-scan, a tokenised form of the input is usually written during the first pass[4].

2.2 Other Indexing Characteristics

1. The dictionary file is an alphabetically sorted list of all indexed words together with pointers into the compressed indexed file. The compressed index entries for a word start on a word boundary and form a contiguous block.

2. Index entries logically comprise `(document-number, offset)` pairs but they are stored as differences rather than as absolute values and the resulting smaller integers are compressed using the Elias gamma-code. The more effective Golomb compression scheme may be used but its compression advantage is offset by the need for an additional pass to determine coding parameters.

3. The use of bit pointers comprising 5 bits for bit-within-word and 27 bits for word-within-compressed-indexfile in hash table entries restricts the compressed index file to a maximum of 512 MB. Unless compression is unusually effective or a more stringent policy is adopted on terms to be indexed, this imposes a restriction on textbase size of approximately 2 gigabytes. This restriction could easily be removed by allocating more space for bit pointers within the hash table but, given current technology, the present design is considered to be a good engineering compromise.

4. During searching, a search of the memory-resident dictionary is used to locate the terms. When searching for whole terms or whole stems this requires a binary search requiring no more than 20 string comparisons of which only the last one or two will require comparison of more than one or two characters. When searching for prefixes, suffixes or substrings, a linear search is used instead. Once the dictionary entry is found, a single disk access suffices to restore the compressed pointers.

[2] The quantity of text data indexed as a single unit is here called a collection.

[3] Different forms of a word, such as `run`, `running` and `runner` are commonly reduced to a single root or stem using a stemming algorithm.

[4] If disk space is severely limited, this can be turned off, with consequent time penalty.

5. To identify the documents containing matches, a search is made of the document table. A binary search is appropriate when small numbers of matches are involved but a linear scan is faster for large match sets. The document table *per se* is relatively small and should be kept in memory to achieve good performance. The list of document titles for a large collection may occupy many tens of megabytes but need not be memory-resident.

6. In practice, if the text of a document is required, it is usually obtained by means external to PADRE (e.g. a remote document server). If obtained by PADRE, the bundle containing the document (or match) is identified by binary search in the bundle table and the bundle is read and decompressed.

2.3 P97 Indexing Algorithm

pass1 Scan raw text input and make bundle table, document table and hash table using a finite-state scanner. Write a tokenised version of the input in which each indexed word occurrence is represented as a `(hash table index, gap since last indexed word)` pair. A complete (bundle-by-bundle) decompression of the input is required. During scanning word frequency data is accumulated as is the number of bits required by each word in the compressed index. The public-domain stemming function used is quite expensive computationally and consequently a second hash table is used to translate raw word variants into stems, ensuring that `Stem()` is called only once for each distinct variant.

pass1 epilogue Scan hash table and calculate index offset and length information for each word and total length of index file. Allocate disk space for the index file and memory-map it.

passes 2, ... Output a fraction of the index in each remaining pass. Each pass writes the complete index entries for all terms whose index offsets lie within the window for this pass. The result should be that writing to the index file is largely restricted to pages in the resident set. The number of passes is determined by the user-specifiable window-size which should be set to a suitable fraction of available physical memory size.

finalisation Sort the hash table into lexicographic order and write dictionary and lexicon.

3 Query Processing Overview

Indexing data structures created using the algorithm described in the section above are used in query processing. The approach to query processing used in PADRE97 is similar to that of earlier versions of PADRE: Matchsets of feature occurrences are created and used to update the relevance score of documents in which they occur; Matchsets may be operated upon to create new matchsets and to update the relevance of documents.

```
topic 253

[ "cryonic suspens"(1)  cryonic(3)
suspens(3) freez(2)  death(2)
futur(1)  practic(1) "quick freez"(1)
bodi(1)  human(1)  preserv(1)
nitrogen(1)  resuscit(1)  cure(1)
feasibl(1)  wealthi(1)  long(1)
"long term"(1)  storag(1)
determin(1) viabl(1) ]

top 1000
```

Figure 2: An example PADRE97 query generated to find documents relevant to the topic: "What is the status of the cryonic suspension industry - background and future prospects?". The `topic` command causes relevance recording structures to be reset and output to be tagged with the identifier "253". The 18 word stems and three stemmed phrases create a single matchset but document relevance is calculated using individual term weights (in parentheses) and individual term frequencies. The `top` command causes the identifiers of the 1000 top ranked documents to be displayed in order of decreasing estimated relevance.

PADRE97 queries are scanned using a *FLEX*-generated scanner. An example of a query is shown in figure 2.

PADRE97 exists in two forms: `P97S` and `P97MP`. The former version operates on single workstations and the latter on clusters of such workstations. In either version, multiple collections may be processed on a single workstation by treating them as a *super collection*. Unlike the earlier PADRE model [?], dictionaries are not merged, reducing the cost of data structure building (and space required) but slightly increasing the cost of term location during query processing. During processing of each term within a query, all collections within the super collection are searched, one after the other, and the results are combined.

3.1 P97MP

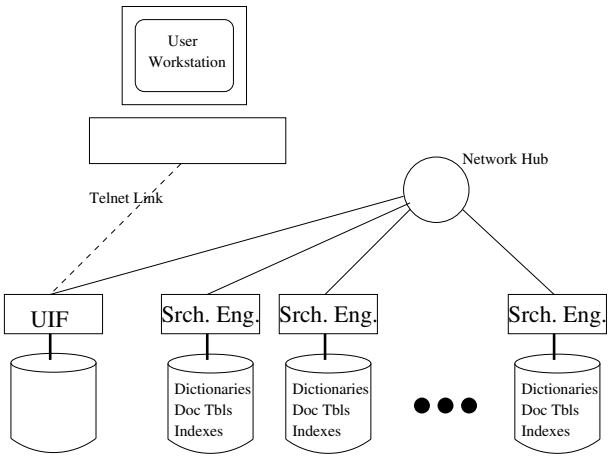

Figure 3: Architecture of `P97MP`. Other variants are supported. In particular, the UIF (User Interface or search manager) may run on the user workstation and the machine running the UIF may also run a search engine, provided that sufficient memory is available and loads are balanced correctly to ensure good performance. Use of the portable MPI system `mpich` and the ability to define new MPI types is expected to allow the use of heterogeneous clusters, but this has not yet been demonstrated.

Figure 3 shows the architecture of the `P97MP` version. Communication between the processes uses the MPI message-passing system [?; ?].

In `P97MP`, the super collections handled by different nodes may comprise different numbers of collections, as illustrated in figure 4. The ensemble of super-collections across nodes is called a super-super-collection.

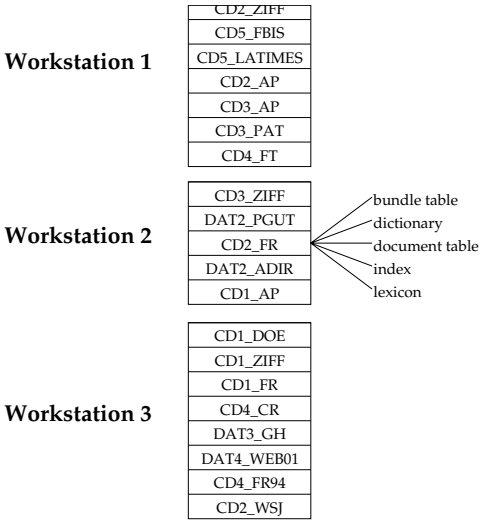

Figure 4: Super-super-collection architecture.

During query processing by this version, communication between workstations is needed in the following circumstances:

1. Multicast of search commands from the UIF to all search engines. The UIF transmits these commands as soon as they are encountered, synchronising with the search engines only when document ranking results are expected from them. In the query shown in figure 2, only three such messages are sent because the term block is pre-processed and sent as a single message.

2. Global reduction between the search engine processes to compute global statistics. This occurs at startup to compute size and number of documents for the entire super-super-collection. Depending upon the relevance scoring method chosen, it may also be necessary to perform global summation of document frequencies for each term, proximity operation or set operation. In the query shown in figure 2, only one such global summation operation is required to compute the document frequencies of all the terms.

3. Merging of ranked document lists. The result of processing a query on a workstation

is a local list of document identifiers in order of decreasing estimated relevance. Each search engine produces its own local ranked list and these are merged by the UIF which requests buffers of such identifiers (and their relevance scores) as needed. In the query shown in figure 2, the UIF process will request buffers of ranked documents immediately after transmitting the terms. However, the search engine processes will not respond until they have finished processing the query. Once buffers have been received from all search engines the UIF merges the rankings, calling for new buffers if needed.

4. Query optimisation based on term frequencies. If local frequencies only are used, different search engines will be working with different term lists.

5. Transmission of text as required by any operations which require display of sections of documents, such as printing the lexicographic context of matches.

6. Internal relevance feedback. This also requires access to document text.

7. Operations which save or restore matchsets, pre-computed document scores or lists of documents.

3.2 Communications Strategy

An excessive number of synchronisation points in a parallel program may significantly reduce performance because of variations in time taken to perform the computation between successive synchronisations. This effect may be magnified on a multi-user COW because of interference from other timesharing programs, especially if gang scheduling is not in operation. Unlike PADRE, PADRE97 requires synchronisation between the processes only when information exchange is unavoidably needed for further computation.

The processing of queries by PADRE97 imposes very little demand on network bandwith. Most data transmitted consists of titles of documents retrieved. Actual processing of even the most complex query should cause traffic of less than a kilobyte . This can be transmitted without significant delay even over Ethernet. However, scalability of query processing may potentially be adversely affected by network latencies on COWs connected by slow networks, or in circumstances where latency is aggravated by the activities of other timesharing users. PADRE97 attempts to minimise this problem by avoiding unnecessary messages and by buffering. If a query comprises 50 literal terms and global document frequencies are needed, the frequencies of all 50 terms are summed in a single MPI operation, thereby reducing the number of buffers transmitted and the number of synchronisation points.

4 Single Workstation Results

Table 1: Comparison of PADRE97 performance on a single workstation with that of PADRE on a large scale parallel machine. Results shown relate to indexing a 2.0 gB collection (TREC CD2/CD4) and using it to process TREC topics 251 - 300. Queries for PADRE were generated manually and are believed to have achieved the second-best result in TREC-5 for non-interactive manual runs. Queries for PADRE97 were generated automatically and were optimised to reduce runtime (with a small loss of effectiveness).

	PADRE on 128node/32disk AP1000	PADRE97 on 256 MB Ultra 170
Perm. disk sp.	6.29 gB	1.30 gB
Temp. disk sp.	0.65 gB	0.32 gB
Time to index	36.3 min.	112.3 min.
Ave. precision	0.2261	0.2255
Time per query	38 sec.	6.54 sec.

Table 1 compares the performance of super-dictionary PADRE on a 128-node, 32 disk AP1000 (as reported in [?]) with PADRE97 on a single Sun Ultra.

As may be seen, PADRE97 is relatively very space efficient and processes queries much faster while achieving almost indistinguishable precision-recall results.

Although indexing times are nearly four times as high for the single workstation, the indexing rate of 1.07 gB per hour elapsed compares favourably with leading retrieval systems. Note that a 1024-node AP3000 system using

PADRE97 software could index text at a rate of more than one terabyte per hour!

5 Multi Workstation Results

?] reported query processing performance results for replicated collections totalling 102 gigabytes on a laboratory of ten Ethernet-connected SGI workstations. These results essentially showed that constant query processing time can be achieved provided that the number of workstations is increased in proportion to the size of the data. However, response time dramatically increased when the same (relatively small-memory) workstation was used to run the UIF process as well as a search engine. The same paper reported good results on a more stringent test of scalability with a 16-node Fujitsu AP+ running AP/Linux.

More recently, experiments have been carried out using a 20 gigabyte test collection [?] (no replication) on an 8-node DEC AlphaFarm interconnected by 155 Mbit/sec ATM. Results are presented in table 2. The data was stored on a RAID array connected to one of the Alphas. Each Alpha was configured with 128 MB of RAM[5], 2 MB of external cache and a 266 MHz processor, but no local disk storage. As may be seen, scalability is less than would be expected on the basis of earlier results, presumably due to the lack of distributed disk storage.

Table 2: Summary of results for indexing the TREC VLC and a 10% sample of it on an 8-node DEC Alpha Farm. Unfortunately, the index building time reported is that for a single node. A time for index building using multiple nodes in parallel is not yet available.

	Sample(2 gB)	VLC (20 gB)	Ratio
Ave. time/query	12.1 sec.	62.7 sec.	5.18
Index bld. time	1.405 hr.	15.6 hr.	11.1
Disk space	1.426 gB	14.06	9.86

[5]192MB on two of the nodes

6 Conclusions

1. The new PADRE97 design is capable of competitive speed and effectiveness on single workstations.

2. Results previously obtained on a Fujitsu AP+ running AP/Linux and on an ethernet-connected cluster of Silicon Graphics workstations suggest that PADRE97 can maintain constant query processing speed as long as the size of the cluster is increased in step with growth in data size.

3. Results reported elsewhere show that PADRE97 is capable of processing queries over collections exceeding 100 gigabytes using as few as ten workstations and a total of only 640 MB of RAM.

4. A cluster configuration with a local disk on each workstation clearly scales better than one in which a parallel disk is connected to only one of the workstations.

5. As predicted at PCW in 1992 [?], it is possible in 1997 to buy a parallel machine (such as the Fujitsu AP3000) with more than one terabyte of RAM. Such a configuration remains unaffordable to the average text searcher! However, PADRE97 results suggest that query processing over terabyte collections could be carried out on a 50-node cluster with a total of as little as 6 gigabytes of RAM.

6. Increasing RAM will improve query processing speed up to the point where all-but very infrequently-accessed dictionary and index entries become memory resident.

7. A number of techniques may significantly speed query processing on a given hardware configuration. Examples include various well-known query optimisations, sorting indexes by term frequency to promote memory-residency, removing position information from indexes, decompressing all or part of the indexes, caching matchpoints or document scores for common queries or subqueries.

8. All stated objectives of the PADRE redesign have been achieved.

9. Further investigation of the relationship between scalability and network latency is needed for large clusters and one-second query processing times.