PADRE — A Parallel Document Retrieval Engine

David Hawking
E-mail: dave@cs.anu.edu.au
Computer Science Department,
Australian National University,
Acton ACT 0200,
Australia
Phone: +61-6-249-3850
Fax: +61-6-249-0010

Abstract

Developments in text retrieval on the AP1000 since last year's PCW are reported. The software, now called PADRE, has been entered in the competition associated with the 1994 Text Retrieval Conference (TREC-3). PADRE is now capable of document relevance estimation and ranking, and supports data loading from and dumping to the Fujitsu Local Filesystem. A new load balancing operation has been devised and implemented and improved techniques for handling cell-program error conditions have been adopted. Experiments have been successfully carried out on document collections exceeding 1.5 million documents and 5 gigabytes of data. Performance results are presented.

1 Introduction

Earlier papers [2] [3] [4] have described the evolution in design of a parallel text retrieval system for the Fujitsu AP1000. The software is now called PADRE. Current work (reported extensively in [6] and [7]) is pursuing the following aims:

1. To add capabilities necessary to retrieve documents relevant to a complex research topic;
2. To experiment with significantly larger datasets;
3. To extend and improve the query language;
4. To improve performance; and
5. To improve robustness

The 1994 Text Retrieval Conference, TREC-3 [1] is preceded by a competition in which participants attempt to use document retrieval systems to retrieve documents relevant to 50 complex research topics from among a collection of two gigabytes of text and of the order of one million documents. The achievement of the above aims was necessary for participation in TREC-3.

2 New Algorithms In PADRE

2.1 Document Relevance Ranking

Being able to quantify the relevance of a document is important in a practical research tool. It enables documents most likely to be relevant to be retrieved first. PADRE bases its estimates of relevance on the following heuristics:

- If a term appears frequently throughout the collection, its presence in a particular document is not particularly significant.
- The more frequently a term appears in a particular document the more likely the document is to be relevant.
- The longer the document, the less weight should be attached to occurrences of search terms. Shorter documents should be ranked ahead of longer documents which would otherwise rank equally.

The above heuristics were used to devise the following relevance measure:

$$R_d = 10000 \times \sum_{t=1}^{k} (I_t \times r_{(t,d)}) \qquad (1)$$

where:

R_d is the estimated relevance of document d,

$r_{(t,d)}$ is the relevance of a document d due to term t,

I_t is the manually assigned importance of term t, and

k is the number of terms.

$$r_{(t,d)} = \frac{f_{(t,d)}}{\sqrt{F_t \times l_d}} \qquad (2)$$

where:

$f_{(t,d)}$ is the frequency of term t in document d,

F_t is the frequency of term t in the entire collection, and

l_d is the length of document d

The square root is introduced to prevent the bias against long documents and against terms appearing frequently throughout the collection from being applied too severely. Other systems have used a logarithmic function. As yet the two alternatives have not been compared.

PADRE introduces a table with a floating point cumulative relevance metric for each document in the collection. Entries in the table are set to zero whenever a change is made to the document collection and whenever a new research topic is signalled. Relevance measure components are calculated for each search term and added to the cumulative metric for each document.

2.2 Returning Documents In Order Of Decreasing Relevance Estimate

The user of PADRE may at any stage request the retrieval or identification of the k most relevant documents to the current topic. These documents are located as follows.

> Each cell finds the highest element in its cumulative relevance measure (`crm`) array. The cells then call `xy_imax()` to find the cell with the global maximum. That cell sends either the text of the document or an identifying label to the host machine. It then zeroes the `crm` entry, saving the old value for later restoration, and scans again for the new maximum entry.

This algorithm is effective but rather inefficient. Investigation of a more efficient alternative is planned.

2.3 Revised Semantics Of Proximity Operators

Earlier versions of the software, (PADDY and FTR) provided **near** and **followed by** operators in which the definition of proximity was a specified number of characters, regardless of document boundaries. This was appropriate when carrying out lexicographic and linguistic searches over a collection in which documents were actually subsidiary parts of a single huge document. However, these semantics are inappropriate when attempting to retrieve documents relevant to a research topic from among a collection of small, independent documents.

Accordingly, PADRE has restricted the scope of the **near** and **followed by** operators to require the component terms to lie within the same document. This restriction avoids the need for communication between cells.

The two operators are now applied to the top n match sets[1] on the match sets stack, where n is specified by the user. This change permits proximity constraints to be applied to match sets resulting from literal searches, regular expressions, set operators, component searches or even other proximity operators. The following example is designed to find occurrences of "Clinton ", "Yeltsin", "strategic arms " and a reference to certain specific types of weapon occurring within 200 characters of each other within the same document.

```
>> {proximity 200}
>> "Clinton "
>> "Yeltsin "
>> "strategic arms"
>> "B1 " + "MX " + "SS-9"
>> near 4
```

2.4 Implementation of Followed By Operator

The implementation of **followed by** is quite simple when $n = 2$. For each matchpoint in the first match set a proximity limit is calculated by adding the currently prevailing proximity range p. If the limit lies beyond the end of the current document, the end of the document becomes the limit.

The second match set is then scanned, looking for a pointer lying between between the current match in the first set and the proximity limit. If such a match is found, the match from the first match set is copied into the result match set. Note that the result set will never contain duplicates, even if more than one pointer in the second set meets the proximity condition.

[1] A match set is an array of pointers, each indicating the first character of a string in the document collection which matched a search pattern.

The procedure is similar for larger values of n. If suitably proximate entries have been found in the first k sets, the $k+1$th set is searched for entries lying between the suitable entry in the kth set and the originally calculated proximity limit. In this way, a match is only added to the result set when all n terms are found in the correct order and with no more than p characters between the start of the first term and the start of the last.

As an efficiency measure in both **near** and **followed by** routines, advantage is taken of the fact that match sets are ordered. A record is kept of safe points in the second and subsequent sets from which to start scanning. The safe point in a set is the highest valued match in that set which comes before the most recently processed match in the first set.

2.5 Implementation of Near Operator

The operation **near** n is carried out as a set sum operation on the results of n operations similar to **followed by** n, looking in each match set in turn for the potential first term of the group. Note that ordering of the subsequent terms in the group is not enforced.

The result set produced by this algorithm consists of n ordered sequences of pointers. To merge these into a single ordered set, the entries are sorted in place using the well-known quicksort algorithm.

3 Larger Data Collections

By loading parts of the TREC-3 document collection twice, an artificial data set big enough to explore the maximum capacity of PADRE on a 512 cell AP1000 was created. During loading of this collection of 5.1 gigabytes, a small number of compressed files failed to load due to lack of memory. These problems could have been deferred by load balancing[2] during loading or by reducing the size of the compressed files. However, these techniques would not increase capacity by much more than 10%.

As reported in [7], performance of PADRE on this large collection was quite pleasing. Literal searches and set union operators took less than two seconds on average, while proximity operators used in typical TREC-3 queries generally completed in less than a tenth of a second. Of greatest concern was the time taken to load from the fileserver over ethernet. Even in compressed form, the data took 87 minutes to load. High performance I/O both from parallel local disks and from external sources is much needed.

[2]In fact, load balancing after initial loading increased the free memory in one cell by approximately 1.4 megabytes.

4 Performance Improvement

4.1 Load Balancing

As described in Hawking and Bailey[4], load balancing is of considerable importance in minimising retrieval times in parallel retrieval software. Experience gained in handling the larger data sets associated with the TREC-3 competition has drawn attention to the importance of load balancing in maximising the size of document collection which may be handled on a given AP1000 configuration. In general, if the maximum amount of memory available for documents on an AP1000 cell is d megabytes, the total quantity D of document data which can be stored on the AP1000 is:

$$D = \frac{N \times d}{LI} \tag{3}$$

where:

N is the number of processing cells, and

LI is the load imbalance (ratio of maximum to average chunk size).

Unless a document collection has been specifically prepared to suit loading on a particular configuration of AP1000, it is not unusual to encounter high values of LI, some in excess of 2.0. As a value of $LI = 2.0$ corresponds to halving the effective memory capacity of the machine and also halving its effective speed, the importance of load balancing is immediately evident.

Hawking [7] describes the method for load balancing now incorporated in PADRE. It is implemented in the form of a user command which may be issued as often as is necessary to achieve the desired result. Though the time to complete a load balancing operation is not excessive, an empirical study has recently been carried out with a view to understanding its behaviour.

First, extra instrumentation was added to PADRE to time separate parts of the load balancing operation and to measure the communication characteristics of the data transfer phase. Each load balancing operation is followed by a PADRE `reset()` in order to recompute the table of document startpoints. The time for the actual load balancing part of the operation was measured as well as the overall execution time for the command. Surprisingly, load balancing proper was observed to take only a small part of the overall time.

The graph in figure 1 shows the relationship between overhead time (elapsed time for the command minus the time for load balancing proper) and the size of the collection. As can be seen, the overhead time is high, even for very small collection sizes.

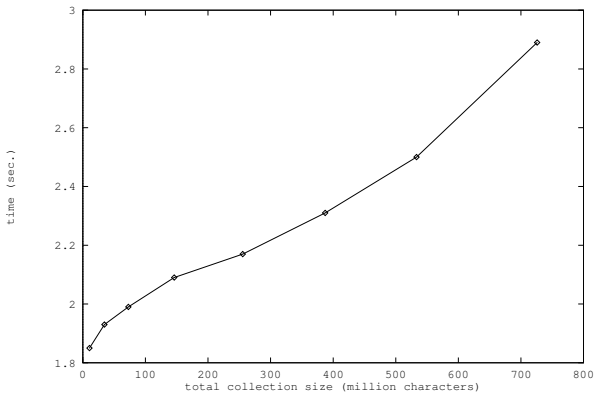

Figure 1: Relationship between the time consumed in the loadbalance "overhead" and the size of the overall collection. No. of AP1000 cells = 128.

There seems to be a time component proportional to the size of the collection and a large constant component. This suggests that the `reset()` operation should be the main target of optimisation. Even if `reset()` can be significantly optimised, load balancing should directly adjust the affected parts of the document start table rather than completely recalculating the whole table by calling `reset()`.

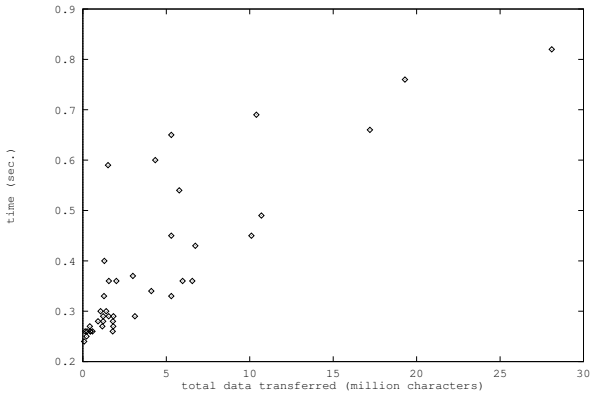

Figure 2: Time taken to balance load plotted against the total amount of data transferred. No. of AP1000 cells = 128.

Figures 2, 3 and 4 plot time for the balancing operation proper against a number of variables. From the plots it can be seen that the total communication time depends most on the size of the maximum individual transfer but there is a small additional influence of total data transferred, presumably due to network contention. A stepwise regression carried out on the data shows that 93% of the variance in time is explainable by the maximum transfer size and 97% by the combination of maximum size and total data transferred.[3] There is no significant correlation between time and number of transfers ($r = 0.05$).

[3]The correlation between the maximum transfer size and the total amount of data transferred is ($r = 0.71$).

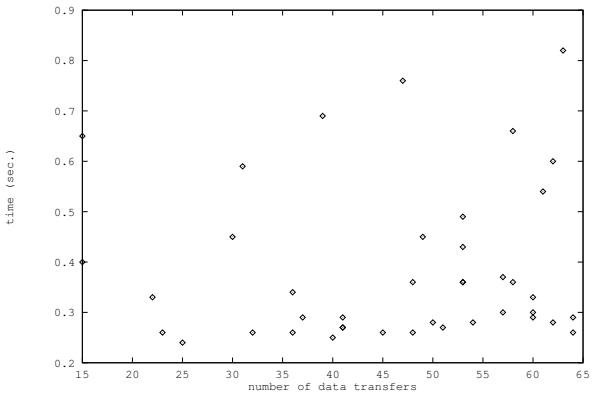

Figure 3: Time to balance load plotted against the number of simultaneous transfers necessary to achieve it. No. of AP1000 cells = 128.

Further work on load balancing could be productively devoted to:

- Reducing the `reset()` overhead;
- Replacing the user loadbalance command with an efficient automatic adaptive multi-pass mechanism;
- Improving the effectiveness of each load balancing pass by allowing the donor cell in a pairwise transfer the freedom to dispose of documents from the middle of its chunk rather than just the tail; and
- Implementing a less-effective but order-preserving variant of load balancing for use when the document collection must remain ordered.

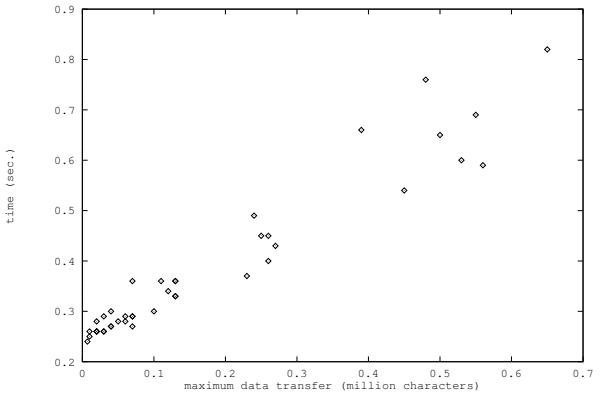

Figure 4: Time to balance load plotted against the maximum amount of data transferred. No. of AP1000 cells = 128.

4.2 Use Of An Inverted File Index To Speed Searching

PADRE is capable of achieving quite acceptable performance using full-text scanning. Two alterna-

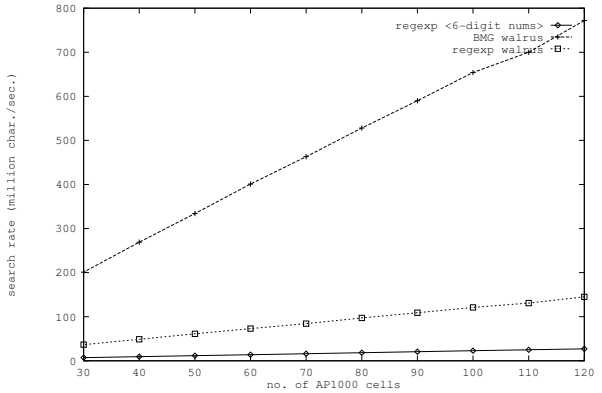

Figure 5: Searching rates for different AP1000 configurations. 286 megabytes of material from the San Jose Mercury were used. Times were measured as elapsed times on the host computer.

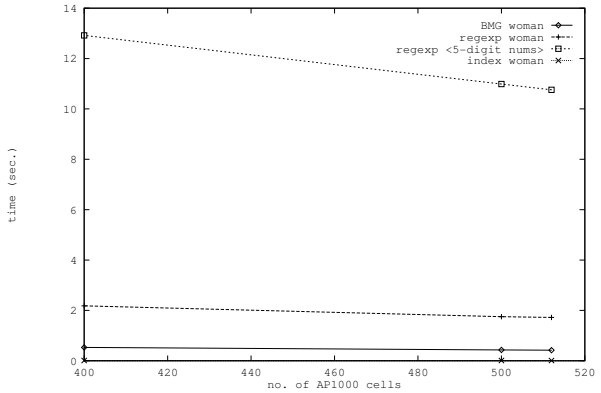

Figure 7: Search times for different AP1000 configurations. Data used was CD1 of the TREC-3 collection (approximately 1 gigabyte). Times were measured as elapsed times on the host computer.

tive algorithms are available: Boyer-Moore-Gosper matching for literal strings and the GNU regular expression code for general regular expressions. Figure 5 shows the searching rates achieved with the two methods. Searching rates are clearly lower for complex regular expressions than for simple ones.

PADRE includes a user command to build an inverted file index. The algorithms for index building and indexed searching have been described in [2] and the time taken to build an index has been analysed in [7]. If an index has been built and the pattern to be searched for is a simple literal anchored at a word start, PADRE will use index-based searching to improve speed.

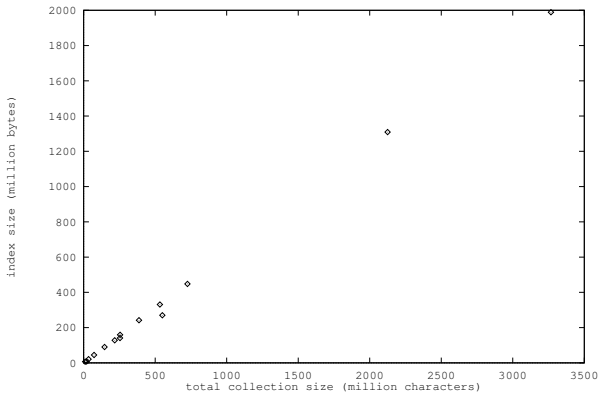

Figure 6: Size of index plotted against the total collection size.

Currently, all words other than SGML tags are indexed. Figure 6 shows a scatter plot of index sizes against the overall collection size. Over the collections shown, the space required for an index ranged from 39% to 63% of the size of the collection, depending upon the amount of SGML markup they contained. The memory requirement of the index could be significantly reduced if *stop words*[4] were excluded from the index.

The rate of indexing a collection on an AP1000 (see [7]) is roughly 2.84 megabytes per minute per cell, corresponding to 1.42 gigabytes per minute on a 512-cell machine.

The effectiveness of indexing in reducing search times is shown in figure 7 which shows the time to locate a literal string over a gigabyte of text using the three different PADRE methods as a function of the number of processing cells. The time taken to process a more complicated regular expression (designed to locate 5-digit numbers) has been included for comparison. The times for indexed search are so low as to be indistinguishable from the baseline. In fact all the points were 0.01 seconds. Unfortunately insufficient time was available to collect data for different numbers of cells.

4.3 Use Of Local Filesystem

The current version of the Fujitsu Local Filesystem provides a very satisfactory model for PADRE I/O to option disks. It provides a level of abstraction in which each cell appears to have its own independent filesystem. When dumping a collection to option disk, each PADRE cell can therefore write its chunk to a file on its own filesystem. An initial version of PADRE which supports dumping and loading via the local filesystem exists but as yet does not provide performance at the desired levels. A future version will hopefully operate at much higher rates and also allow dumping and loading of compressed files.

The independent filesystem model makes the representation of a document collection highly dependent on the number of cells in the particular AP1000. However, this is not a serious limitation in practice

[4] Stop words are words with little information content such as "to", "the", "and", etc.

as the disks are connected to only one AP1000 and cannot be accessed by another AP1000 except via the host machine. Unless timing experiments are being performed, PADRE always uses all available cells.

5 TREC-3 Participation

The TREC research topics are generally quite elaborate and result in highly complex PADRE queries. The manually generated queries in the ANU entry averaged 27 literal terms and 25 set or proximity operators. Multiplied by 50 topics, this generated a computational load extending for approximately 50 minutes on the 512-cell AP1000, not including data loading.

Error-free runs of this magnitude over two gigabytes of data required a high degree of program robustness. The techniques employed for error handling and behaviour validation are described in [7]. In addition, the PADRE code was improved to avoid memory leaks, to test for table overflows, to impose a discipline of message passing designed to avoid unsatisfiable memory demands and to check return codes of message passing and memory allocation routines. While still not complete, these improvements did result in apparently error-free runs.

The following is the text of one of the TREC-3 topics:

> <title> Topic: Oil Spills
>
> <desc> Description:
>
> Document will identify major oil spills throughout the world.
>
> <narr> Narrative:
>
> A relevant document will note the location of the spill, amount of oil spilled, and the responsible corporation, if known. This will include shipborne accidents, offshore drilling and holding tank spills, but should not include intentional spills such as Iraq/Kuwait or leakage from broken pipes. References to legislation brought about by a spill, litigation and clean up efforts associated with a spill are not relevant unless specifics of the spill are included.

The manually generated PADRE query was as follows:

```
topic 154
{proximity 100}
{weight 0}
"oil " + "crude "  + "hydrocarbon"
"spill " + "release " + "accident"
+ "discharge"
"harbour " + "sound " + "sea " + "ocean "
+ "seaway " + "lake " + "river "
+ "canal " + "waterway " + "water "
+ "bay " + "estuary "
{weight 1000}
near 3
{weight 0}
"Saddam " + "Iraq"
{weight -990}
near 2
{weight 0}
"oil " + "crude "  + "hydrocarbon"
+ "discharge"
"spill " + "release " + "accident"
"harbour " + "sound " + "sea " + "ocean "
+ "seaway " + "lake " + "river " +"can
al " + "waterway " + "water " + "bay "
+ "estuary "
"gallons " + "tons " + "tonnes "
+ "barrels " + "slick "
{weight 1000}
near 4
top 1000
```

The manually generated queries contained many instances in which a search was made for one of a series of alternative words or phrases. This was achieved by searching for each of them individually using the Boyer- Moore-Gosper (BMG) algorithm, then combining the result sets using set union + operators. When there are more than a few alternates, it is likely that a finite state automaton approach could be used to search for all of them in a single pass through the text. This approach is likely to be investigated in the future.

In addition to manually generated queries, software developed by Paul Thistlewaite was used to generate PADRE queries automatically from the TREC-3 topics.

6 Discussion And Conclusions

Results from the TREC competition have not yet been thoroughly analysed but the major goals of participating in TREC have already been achieved.

- PADRE has been shown to be capable of efficiently processing complex research topics over document collections of up to 5 gigabytes on a 512-cell AP1000.
- PADRE has been shown to be able to rapidly respond to changes in such a document collection. (See [7])
- PADRE supports powerful search terms such as regular expressions which are not easily supported by conventional systems. (See [6])

- The PADRE framework is able to support efficient editing operations over large collections.

It is confidently expected that the data handling capacity and speed of PADRE will scale up well with proposed increases in cell speed and memory capacity (AP1000+). PADRE performance also scales well with increasing numbers of processing cells. However, efficient I/O is essential to make use of increased processing capacity in this application.

Future work on PADRE is likely to include query language and user interface improvements, optimisation and experimentation with different types of data.

Acknowledgments

My thanks to Paul Thistlewaite for providing the automatically generated queries for the TREC-3 participation and to Robin Stanton for continued support and advice. I am also very grateful to FPCRF for access to the 512-cell AP1000. Thanks also to Peter Bailey and Michael Hiron for past work on elements of the PADRE code. Finally, thanks to NIST in the USA and various copyright holders for access to the TIPSTER data collection.

References

[1] Harman, D.K. *The Second Text REtrieval Conference (TREC-2)*, US Department of Commerce, NIST Special Publication 500-215, (Mar 1994).

[2] Hawking, D.A. "High Speed Search of Large Text Bases On the Fujitsu Cellular Array Processor," *Proceedings of the Fourth Australian Supercomputing Conference* pp. 83-90. Gold Coast, Australia, (Dec 1991).

[3] Hawking, D.A. "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* paper ANU-8, Kawasaki, Japan, (Nov 1992).

[4] Hawking, D.A. and Bailey, P. "Towards a Practical Information Retrieval System For The Fujitsu AP1000)," in *Proceedings of the Second Fujitsu Parallel Computing Workshop* paper P1-S, Kawasaki, Japan, (Nov 1992).

[5] Hawking, D.A. *PADRE User Manual* Department of Computer Science, Australian National University, Canberra, Australia, Oct 1994.

[6] Hawking, D.A. "Searching For Meaning With The Help Of A PADRE," *In preparation for the 1994 Text Retrieval Conference, Gaithersburg USA (November 1994)*

[7] Hawking, D.A. "The Design And Implementation Of A Parallel Document Retrieval Engine," *In Preparation*