Journal of the American Society for Information Science

J A S I S

VOLUME 48 NUMBER 10 OCTOBER 1997

CONTENTS

(Continued on next page)

Cover: Cosmos and chaos-is there any hope of real order? Clustering, classification. indexing!-Adrienne Weiss, Designer. Inset illustration: Andrzej Dudtinski

Clustering and Classification of Large Document Bases in a Parallel Environment

Anthony S. Ruocco
Department of Electrical Engineering and Computer Science, United States Military Academy West Point, NY 10996. E-mail: ruocco@eecsl.eecs.usma.edu

Ophir Frieder'
Department of Computer Science, Florida Institute of Technology Melbourne, FL 32901. E-mail: ophir@cs.gmu.edu

Development of cluster-based search systems has been hampered by prohibitive times involved in clustering large document sets. Once completed, maintaining cluster organizations is difficult in dynamic file environments. We propose the use of parallel computing systems to overcome the computationally intense clustering process. Two operations are examined. The first is clustering a document set and the second is classifying the document set. A subset of the TIPSTER corpus, specifically, articles from the Wall Street Journal, is used. Document set classification was performed without the large storage requirement (potentially as high as 522M) for ancillary data matrices. In all cases, the time performance of the parallel system was an improvement over sequential system times, and produced the same clustering and classification scheme. Some results show near linear speed up in higher threshold clustering applications.

Introduction

The growth of electronically available information is staggering. Magazines and newspapers are available in electronic mediums. Formal correspondence, once delivered as a physical entity (hard copy), can now be sent electronically from one computer directly to another. E-mail has taken on a significant role as a means of correspondence. A paper trail in today's environment may, in actuality, be a series of electronic correspondence. With the growth of electronic text-based information increasing, information retrieval systems must be prepared to process large amounts of data. As systems become inundated with more and more information, it may not be possible for people to fully understand what they have collected. The ability to produce information that categorizes the data, that is, the ability to produce metadata, is in many cases as important as identifying specific pieces of data within a document set.

The ever-increasing size, coupled with the increasing requirements to classify, group, and process the document sets, all within nonprohibitive execution times, motivates the use of parallel processing computers. Parallel information retrieval focuses on this particular domain (Pogue, 1988; Rasmussen, 1991; Reddaway, 1991). Query processing assumes an organized data set as input. We, however, rely on parallel computing to organize the data by performing two cluster preprocessing operations. The first operation is clustering the document set, The second operation is *classifying* the document set. Each operation will be discussed in depth along with a presentation of results attained using a subset of the TIPSTER corpus, specifically, articles of the Wall Street Journal from 1987, 1988, and 1989. For each parallel operation, the resulting document organization is the same as that created by its respective sequential processes.

Previous work in parallel environments focused on the SIMD model (Olsen, 1995; Pogue, 1988, Rasmussen & Wilet, 1989; Reddaway, 1991; Willet, 1988). We performed our studies based on the MIMD model. Specifically, our work was done on an Intel Paragon. This architecture is a 2-D mesh of nodes. Each of the 352 GP nodes consists of two Intel i/860XP processors and 32M of memory. Effects of message passing and file sharing under the MIMD model are also discussed.

*Ophir Frieder is currently on leave from the Department of Computer Science. George Mason University, Fairfax. VA.

Received April 3, 1996; revised December 4. 1996: accepted December 4. 1996.

Parallel Clustering

Clustering is a means of placing significant amounts of data into a comprehensible number of categories (Anderberg, 1973). Clustering methods are chosen based on their general acceptability within an application area (Dahlhaus, 1992; Jain & Dubes, 1988; Rasmussen & Willet, 1989; Willet, 1988). In the area of information retrieval, clustering schemes based on the vector-space mode! are the acceptable norm.

Vector Formation

The document vector, comprised of T unique terms, is the only representation of a document used in the clustering process. Consider a document set consisting of three terms A, B, Γ. Document D_x has a vector representation of $\langle \alpha, \beta, \gamma \rangle$, with α the weight of occurrences of A, β the weight of occurrences of B, and γ the weight of occurrences of Γ. It is possible to create a binary vector, in which case α, β, γ would take on values of 1 or 0. However, approaches using weighing schemes have shown such marked performance increases that their use is preferred (Kwok, 1990). Within the scope of this article, term weights were based strictly on the appearances of words within a document.

A cluster vector has the same format as a document vector. That is. a series of term weights. The net result is that the cluster vector represents the mathematical average of all the documents that comprise that cluster.

Cluster Formation

The single-pass method is one of many methods for forming clusters within the field of information retrieval (Salton, 1989). It is a rather simple approach for clustering. Basically, a document is compared to all existing clusters in turn. If a threshold for a measure of similarity is exceeded, the document is added to that cluster. If a document cannot be placed in a cluster, it forms a new cluster against which subsequent documents will be compared as well. Because clusters are examined in the order they were formed, the early clusters tend to be larger than clusters formed later in the process. Also, the clusters formed are based on the order in which documents are accessed. Thus, the composition of any given cluster, and even the number of clusters themselves, is order dependent.

For a comparison to be made for the single-pass method in a parallel environment, it is critical to maintain both the order in which documents are accessed and to ensure the documents are compared and placed in clusters in the same order as in a sequential environment. These constraints are addressed in detail in later sections.

Before the clustering process begins, there must be a decision on when two documents are considered similar. This is done through a measure of similarity. If a predetermined threshold is exceeded, then the document'and cluster are considered similar. The document gets placed in the cluster, and the cluster vector is adjusted as needed. While there are many measures of similarity associated with clustering, there is no single best method (Salton, 1989). However, one of the more popular, and the one we used, is the Cosine Coefficient, shown below:

$$\mathrm{COSINE}(C_j, Y) = \frac{\sum c_{jj} y_i}{\sqrt{\sum c_{jj}^2 \sum y_i^2}}$$

$c_{j,i}$ = Weight of the ith term of cluster C_j

y_i = Weight of the ith term of document Y (1)

Initially, there are no clusters. The first document, D_1 by default, becomes the first cluster, C_1. The second document, D_2, is compared using the cosine coefficient. If the threshold is exceeded, i.e., COSINE $(C_1, D_2) \geq$ (', D_2 is placed in cluster C_1, otherwise it forms a new cluster, C_2. The next document is compared to C_1 and C_2, where it either is placed in one of the two clusters or forms its own. This process continues for each document.

Parallel processing provides an opportunity to compare a document to more than one cluster at a time. If C_1 and C_2 are on separate processors, the incoming document can be compared to C_1 and C_2 simultaneously. It is possible to compare each incoming document to as many as P *clusters*, where P is the number of processors in the system. P is finite in practice, so it is necessary to apportion the clusters among processors. This is done in a round-robin fashion. Thus, the number of clusters assigned to any given processor can be at most one greater than the number of clusters assigned to any other processor.

A two-phased message scheme ensures documents are properly accessed and placed. During phase one, the processors are primarily concerned with comparing a document to their clusters. Once it finds a cluster that exceeds the threshold, it signals the other processors. Once leaving phase one, a processor cannot be signaled with the same type message. A second message is needed. This second message serves two functions. First, it is an arbitrator. It takes a value of a potential cluster from each processor and determines which processor found the lowest numbered cluster. The designated processor modifies or creates a cluster vector accordingly. The second major function of the phase two message is to serve as a synchronizing point to ensure a decision is made on each document before the next document is processed.

Vector Implementation

A document vector must account for each of the T terms in the document set. A document vector is a series of tuples. The first element of the tuple represents the weight of a unique term, and the second element of the tuple represents the administrative number of that unique term. To facilitate subsequent operations, the first tuple in the document vector has a special meaning. The first element of this tuple represents the total number of words appearing in the document. The second element of the tuple represents the total number of unique terms within the document vector.

The cluster vector also needs to account for each of the T unique terms. It uses the same format as the document vectors. The first element of the tuple represents the weight of a term within the cluster, and the second element of the tuple is the administrative number of that unique term. The first tuple of the cluster vector, like the first tuple of the document vector, provides information on the total number of words within the cluster and the total number of unique terms within the cluster. To minimize overhead, the clusters were physically implemented as linked lists. Each element of the linked list is a record representing the two element tuple.

The physical implementation of the document vectors is more complex. The primary concern was that each processor must access the same files at the same time. This lead to significant problems with file I/O. It is known that file I/O can be a system bottleneck and that performance for I/O is highly system dependent (de Rosario & Choudhary, 1994). There are several other concerns with file I/O, such as system load, over which the user has very little control.

The Intel Paragon offers several methods for file access. These are categorized based on the level of synchronization inherent in each mode. A detailed description of the performance characteristics of each mode is provided by Nastea and colleagues (1996). The M_UNIX mode, which provides each processor with a unique file identifier for shared files, along with a block size of 64K, was selected.

The individual document vectors were packed into a single file such that no document was split across file blocks. The first tuple of each document indicates how many tuples were in that vector, that is, the number of unique words. Each document vector begins at some offset from the beginning of its block. This offset is simply the sum of the header tuples of the previous documents. Block reading is done asynchronously. While the last vector of a block is processed, the next block is read.

The above details and considerations are not unique to this application. In fact, they are implementation rather than application oriented. However, in the context under which the research was performed, such details are important. Many times, when issues dealing with processing data are examined, the time to read that data into the system is considered problematic. That is, it is so system and load dependent that its description detracts from the processes in question. However, the computing environment was based on strictly enforced run-time limits. The time to read data into the system subtracts from the available time the processors can perform the clustering process. Therefore, the time to have data enter the system is not problematic, but integrally embedded into the overall process. Isolating reading from clustering was not possible.

Results

The experiments were run under several parameters. These parameters are:
(a) $\mathcal{D}$, the number of documents in the document set; (b) $\mathcal{O}$, the order in which the document vectors are accessed; (c) $\mathcal{C}$, the threshold value for the Cosine Coefficient; and (d) $\mathcal{P}$, the number of processors in the system.
A permutation is defined using the notation [$\mathcal{D}$, $\mathcal{O}$, $\mathcal{C}$, $\mathcal{P}$].

There are two orders represented. The first is a random ordering (R) of the document set. The second ordering is based on the number of unique words (UA) per vector. In other words, the documents were arranged in ascending order such that the first document accessed had the least number of unique words, and the last document accessed had the most unique words. Three thresholds, 0.2, 0.5, and 0.8, were selected for the Cosine Coefficient. These were selected to give a range of clusters formed under the process. In general, the higher the coefficient threshold, the larger the numbers of clusters formed, and those clusters will be tighter. Some document sets were too large to be executed on one or two processors within allocated system time constraints. For most combinations, increasing the number of processors stopped when it was evident that performance, based on speed up, was not improving. Times listed in all tables, and depicted on all graphs, are in seconds and include message-passing and file-reading time.

Cluster Data. Table 1 provides some statistics of the clusters formed under the process. Because the resulting clustering scheme produced by the parallel single-pass algorithm is the same as that created by a sequential version of the algorithm, detailed analysis of the composition of the clusters was not performed. The general pattern is similar regardless of document set size and order. As the coefficient increases, the numbers of clusters increases. The single-pass method "tends to result" in uneven sized clusters. The standard deviation of the average number of documents per clusters highlights this tendency. As seen by the table, the tendency for uneven clusters appears to be related to the document order as well as the threshold value of the coefficient. When examining the UA data, the clusters tend to be larger, but the standard deviation of those clusters indicates they are more evenly sized.

Run Data. Table 2 depicts the times and speed up attained when executing the program on the 5000 and 10,000 document vector, random set. The number of proc-

TABLE 1. Cluster data.

	[5000, R, C, x]			[5000, UA, C, x]		
	0.2	0.5	0.8	0.2	0.5	0.8
Clusters	631	3564	4884	621	3545	4888
Largest	2215	288	21	1601	273	21
Avg Size	7.92	1.40	1.02	8.05	1.41	1.02
SD	90.73	5.46	0.41	76.15	5.19	0.41
	[10000, R, C, x]			[1000, UA, C, x]		
	0.2	0.5	0.8	0.2	0.5	0.8
Clusters	930	6421	9674	933	6397	9672
Largest	4744	566	40	2775	543	40
Avg Size	10.75	1.56	1.03	10.72	1.56	1.03
SD	159.07	8.36	0.64	115.62	7.99	0.69

essors varied based on the coefficient. For a coefficient of 0.2 the maximum processors used was 8, but 16 were used for the higher coefficients, with 32 processors used on the 10,000 document set. The first striking feature of the table is the increase in time to perform clustering as the coefficient increases. The previous section showed how the increase in coefficient significantly changed the number of clusters. Table 2 shows the significant efforts that must be expended to produce those additional clusters.

There are several aspects that must be explored when reviewing performance. Chief among them is the effects of message passing. This application uses a great deal of message passing, as it must ensure consistency of clusters with the sequential version. There is a base number of messages that must be sent, that is, the number of documents, D. So whether the application is run at 0.2, 0.5, or 0.8, there must be a minimum of D messages. These are the phase-two messages described previously. The formation of a new cluster generates a phase-one message. So at higher coefficients, there is actually a larger number of messages being sent. While message passing is a factor, it cannot arbitrarily be considered the dominant factor in performance. Instead, what needs to be considered is the amount of time the processors are working between messages. At the lower coefficients, there are fewer clusters to check. As these few clusters get distributed among processors, there is in effect less work being done and a greater proportional amount of time spent in message passing. Therefore, the ability to scale to more processors is limited. As the numbers of clusters increases, each processor becomes responsible for a greater number of clusters. Each processor is doing more work in checking clusters, so that the amount of time between messages is increased. The effect of doing more work between messages is indicated by the better speed-up performance. Figure 1 graphically depicts the times and speed up attained.

As part of the overall experiment, the document set was specifically ordered based on the number of unique

TABLE 2. Times and speed up for (a) [5000, R, C, P] (b) [10000, R, C, P].

	0.2		0.5		0.8	
P	Time	Speed up	Time	Speed up	Time	Speed up
a						
1	2411.7	1	15552.8	1	24008.5	1
2	1823.7	1.32	8766	1.77	12211.4	1.97
4	1635.2	1.47	4826	3.22	6215.5	3.86
8	1396.6	1.73	2566.1	6.06	3185.3	7.54
12			1841.5	8.45	2157.9	11.13
16			1494.1	10.41	1687.8	14.22
b						
1	6882.1	1	53548.7	1		
2	5449.0	1.26	30578.4	1.75	48287.9	2
4	5288.8	1.30	18323.4	2.92	24469.3	3.95
8	5274.4	1.30	10454.4	5.12	12550.4	7.69
16			5875.8	9.11	6389.8	15.11
32			3631.9	14.74	3696.4	26.13

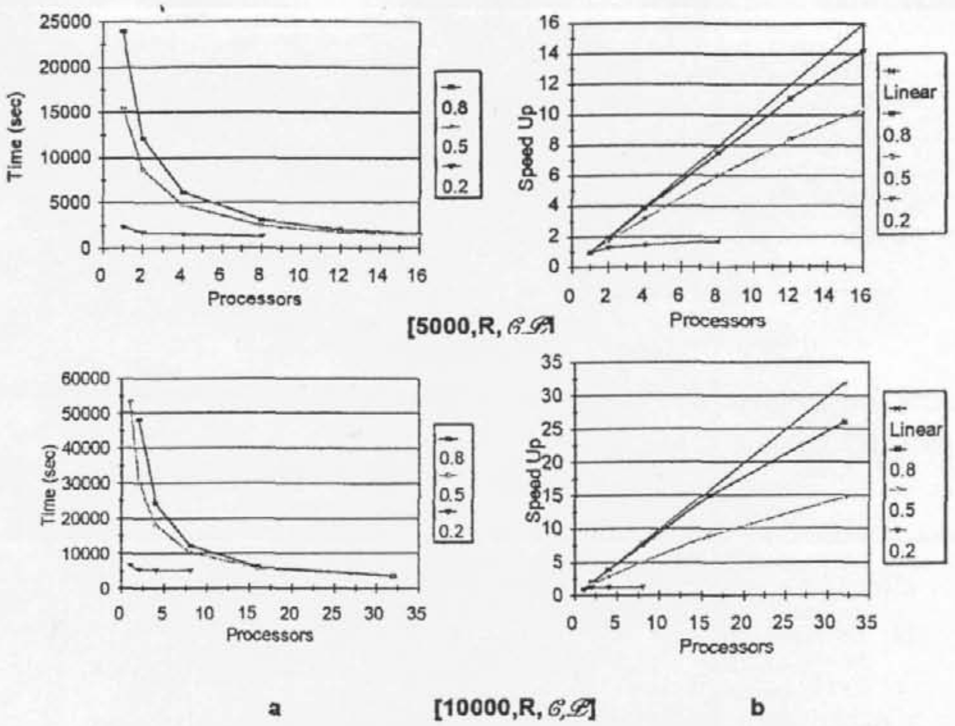

FIG. 1. Time (a) and speed up (b) for the R sets.

words in each vector. This set was based on the number of unique words in ascending order, or UA. The pattern of the data in Table 3 and Table 2 appears to be the same. That is performance in terms of speed up improves as the coefficient increases. The explanation of this directly follows from the discussion above. The relatively few clusters at the lower coefficient is a limiting factor. That is, the amount of work done by the processors between messages is small and scales poorly. As the number of clusters increase, the work load stays at a level that promotes scalability. Eventually, though, this time between messages diminishes and the message passing becomes the limiting factor. This is noted at the higher coefficients.

What is more interesting is a comparison between the sets themselves. Again, as there was no intention of determining "better" clusters, the comparison is based strictly on run-time performance. [Cluster quality studies appear in the literature (Anderberg, 1973; Jain & Dubes, 1988; Li, 1990; Salton, 1989).] To show a comparison between this ordering and the random ordering, the time of execution graphs are superimposed in Figures 2a, b, and c. In almost all instances, the UA run had better times. This is especially noticed at a coefficient of 0.2 (Fig. 2).

By examining the cluster data table (Table 1), it is clear that the UA clusters themselves are better balanced. That is, there is, in general, a smaller standard deviation among numbers of document vectors per cluster in the UA set. The smaller standard deviation implies that each cluster is closer to the same size, and this means that the processors are doing the same amount of work to maintain the clusters. In the situation of the R set, the clusters vary greatly in size. Thus, those processors assigned larger clusters are responsible for a greater share of work. The net effect is the UA set promotes a more work-load balanced environment than the R set. When there are few clusters, this imbalance is more evident; hence, the larger difference in execution times at the 0.2 threshold. While

TABLE 3. Times and speed up for [5000, UA, $\mathcal{C}$, $\mathcal{P}$].

	0.2		0.5		0.8	
P	Time	Speed up	Time	Speed up	Time	Speed up
1	1652.3	1.00	114116.8	1.00	21314.2	1.00
2	1362.4	1.21	7503.6	1.88	10697.7	1.99
4	1096.0	1.51	4365.1	3.23	5368.3	3.97
8	1087.5	1.52	2557.5	5.59	2714.1	7.85
12			1969.5	7.17	1865.1	11.43

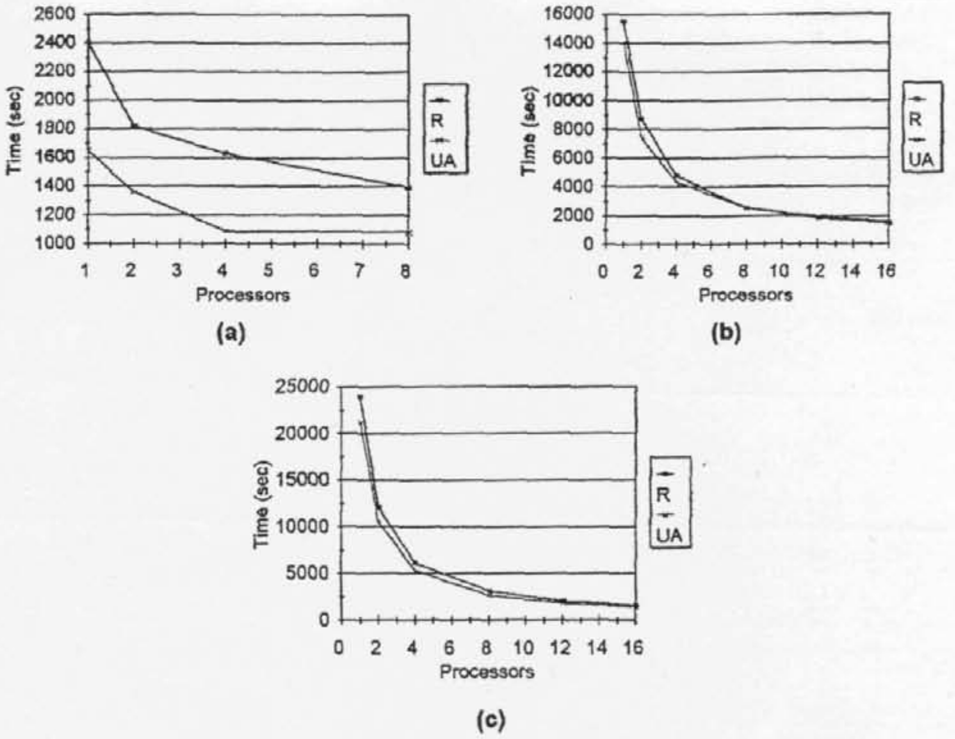

FIG. 2. Time comparison for R, UA at (a) 0.2 (b) 0.5 (c) 0.8.

this was an interesting note, the order effects on cluster formation is not further examined.

Parallel Classification

The goal of classification is to provide some insight into the organization of the data themselves. Unlike clustering, which typically supports a larger system, the classification scheme is the end goal of the process. Classification is an agglomerative process. That is, the leaves contain the most specific information, and each interior node provides more and more general information until all information is consolidated into one group.

The basis of classification is to reflect some measure of resemblance among the items being classified. The resemblance can be based on how similar items are, that is using a similarity measure, or the resemblance can be based on how different the items are, hence, a dissimilarity measure. As Romesburg states, the option of using similarity measures over dissimilarity measures is "a matter of direction" (Romesburg, 1984). In other words, items with a high similarity measure are more similar than items with a low value for the similarity measure. Conversely, items with a low value of dissimilarity are more similar than items with a high value for a dissimilarity measure.

In the area of information retrieval, a classification scheme resulting in a dendrogram can be useful to the analyst. It offers a view of the data that indicates the level of similarity among the documents. The ability to represent the data themselves supports the role of the dendrogram as a form of metadata. To illustrate the use of a dendrogram as metadata, an example is provided. This example consists of a five-document set. The resemblance of each document to every other document in the set is known. Using these measures, it is possible to determine a classification scheme, and subsequent dendrogram for this set. The details of the measures used and how the dendrogram is computed is detailed in subsequent sections. Figure 4 portrays a dissimilarity matrix, which is a square matrix, and resulting dendrogram for the document set.

To read the dendrogram, the documents, identified by an administrative number, appear across the top. A vertical line from the document represents the dissimilarity of the document to the document immediately to the right. For example, the level of dissimilarity between document 1 and 2 is .1 and the level of dissimilarity between document 4 and 3 is .2. The lower the value, the less dissimilar its measure (and by convention, the more similar the documents). The unending vertical line from document 3 reflects the fact that document 3 is the last document in the set. The order of the documents and the resulting dissimilarity are obtained as a result of the algorithm being employed.

The chart is now entered from the left side. A horizontal line indicates the level of dissimilarity among the documents. By entering the figure at the various levels,

	1	2	3	4	5
1	-	.1	.9	.6	.4
2	.1	-	.7	.5	.3
3	.9	.7	-	.2	.6
4	.6	.5	.2	-	.4
5	.4	.3	.6	.4	-

a. Dissimilarity Matrix

b. Dendrogram

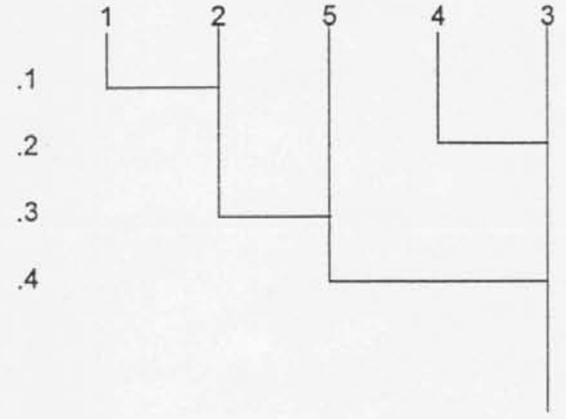

FIG. 3. (a) Sample dissimilarity matrix and (b) dendrogram.

it is evident that the most similar documents are {1,2} followed by {4,3}. The consolidated group {1,2} can the be grouped with document 5 to form group {1,2,5}. At the 0.3 level there are two groups, {1,2,5} and {4,3}. The entire document set gets consolidated at the 0.4 level.

Classification Process

There are numerous classification methods, but the four most common are the unweighted pair-group method using arithmetic averages (UPGMA), the single link method (SLINK), the complete link method (CLINK), and the Wards minimum variance method. There is no categorical best method (Anderberg, 1973; Jain & Dubes, 1988; Romesburg, 1984; Spath, 1982; Willet, 1988).

With little definitive theory on selecting one method over another, the choice for SLINK is application based (Willet, 1988). A document set may not have any related documents, yet the classification methods will still produce a classification scheme. Wards method, with its reliance on sums of squares, imposes an a priori restriction into the document set. This a priori restriction is that there is an inherent norm to the data set that the sums of squares are measuring against. The complete link and UPGMA produce dendrograms which imply a relationship among documents that can be misleading. The SLINK algorithm, on the other hand, exhibits a characteristic known as chaining. As new clusters form, they bring in the dissimilar documents and form a distinct pattern. Figure 4 depicts the chaining property exhibited by the SLINK approach. Part (a) shows how six elements, consisting of two attributes, are dispersed in a two-dimensional plot; the dendrograms have had numbers removed for clarity. In part (b), the single link method shows how first item 1 and 2 are combined, then 3, then 4, then 5, and finally 6. The chaining effect is seen in the dendrogram. Parts (c) and (d) were produced by the complete link and UPGMA

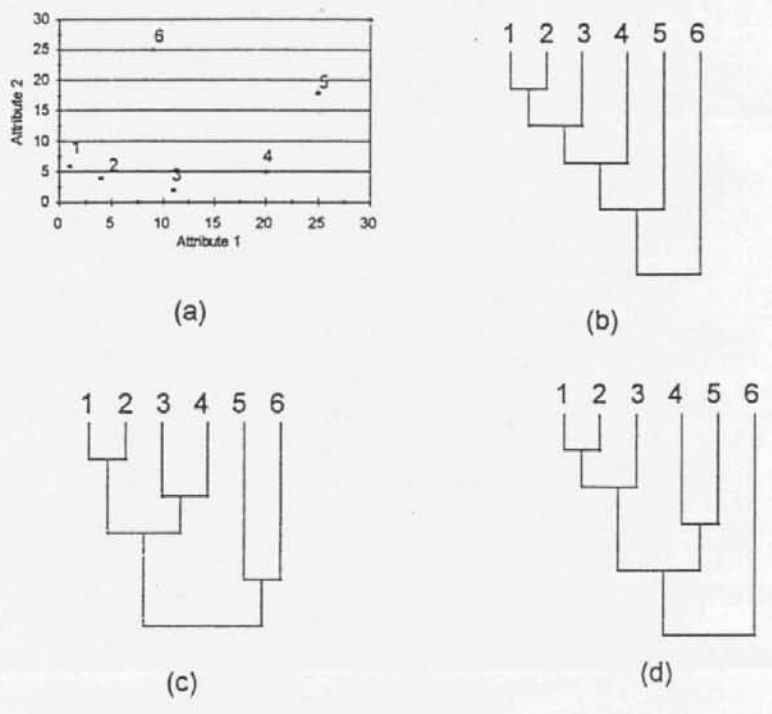

FIG. 4. Example of chaining.

methods, respectively. They imply closer relationships among the documents than is truly present.

Many classification methods rely on a dissimilarity matrix. It is a square, symmetric matrix. Each element of the matrix $[r, c]$ represents the dissimilarity between item r and item c. In the case of document classification, each $[r, c]$ element is the dissimilarity between document r and document c. Element $[r, r]$ has no meaning.

In most references on classification, the dissimilarity matrix already exists and is readily available. However, this belies several important factors concerning the dissimilarity matrix. First, the matrix itself can be large. Basically, the number of elements in a dissimilarity matrix is D^2, where D is the number of documents.

Producing the matrix can also be a drain on resources. The matrix represents the dissimilarity of each document to every other document. This implies a process on the scale of D^2, where D is again the number of documents. If these values are real numbers, and the underlying system uses 6 bytes per real number, the matrix for a 5000 document set would require approximately 143 M of space. The 10,000 document set matrix would require approximately 522 M of space.

Single-Link Implementation

The single-link method is often referred to as the nearest neighbor classification method. Its reliance on selecting the closest neighbor makes it suitable for implementation with minimum spanning tree methods (Dahlhaus, 1992; Olson, 1995; Rohlf, 1982; Sibson, 1973). There are three dominant minimum span tree algorithms in the literature; these are by Solin, Kruskal, and Prim (Kumar, Grama, Gupta, & Karypis, 1994; Quinn, 1994). The choice of one over the other typically relies on the number of edges present in the graph. If the dissimilarity matrix is likened to the proximity matrix of graph algorithms, then it is evident the graph is completely connected. That is, every vertex (document) is connected to every other vertex (document). In the case of dense proximity matrices, Prim's algorithm is generally preferred as it is easier to implement (Aho, Hopcraft, & Ullman, 1983). The algorithm is iterative in nature. It begins with an empty tree then adds vertices, always picking the nearest vertex, until all vertices are included in the tree. Prim's algorithm utilizes two matrices, a dissimilarity matrix, and a distance matrix.

The distance matrix is used to identify the nearest neighbor at any given iteration. The ith element of the distance matrix logically represent the ith document in the set. That is, element 1 represents document 1, element 2 represent document 2, etc. The value in the element represents the distance to the nearest neighbor of ith element. At each iteration, the element with the smallest value gets added to the tree. An illustrative description of

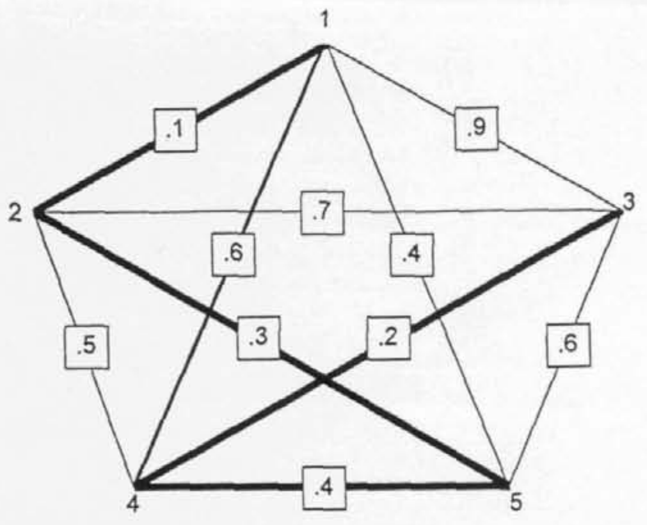

FIG. 5. Sample graph and tree.

the algorithm utilizing the dissimilarity matrix of Figure 4 is provided.

Initially, all values in the distance matrix are large. The process begins with document 1. The dissimilarity between document 1 and every other document is determined. If the resulting dissimilarity from document 1 to document 2 is less than the value in the 2nd element of the distance matrix, the value in the distance matrix is replaced with the smaller value. This continues for each dissimilarity from document 1. At this point the distance matrix would contain $d[1] = -$, $d[2] = .1$, $d[3] = .9$, $d[4] = .6$, $d[5] = .4$ (note $d[1]$ has no meaning as that document is already in the tree). Having the smallest value, document 2 is added to the tree.

Document 2 is now compared to all other documents. It does not get compared to document 1. When document 2 is compared to document 3, its dissimilarity is less than the value in the distance matrix for $d[3]$; therefore, the value of .9 is replaced with .7. Document 2 is compared to the other documents with the distance matrix becoming $d[1] = -$, $d[2] = -$, $d[3] = .7$, $d[4] = .5$, $d[5] = .3$. Document 5 is selected because it has the lowest value in the distance matrix. Document 5 is compared to the remaining documents with the distance matrix becoming $d[3] = .6$ and $d[4] = .4$. Document 4 is selected and is compared to document 3, which changes the distance matrix so that $d[3] = .2$. Document 3 is the last document of the set; hence, it is added to the tree. The resulting tree is shown in Figure 5.

It is clear the algorithm does not need the dissimilarity matrix in its entirety. In fact, the algorithm never accesses more than one element of the dissimilarity matrix at any given time. Once that element is used, it is never used again. The algorithm merely needs to keep track of the element with the best, for example, smallest value, and that is done through the distance matrix. Therefore, there is no need to have the dissimilarity matrix. Instead, it is necessary to have the ability to compute the elements as needed. As explained in previous sections, if the dissimilarity matrix is not used elsewhere, its absence results in substantial resource savings.

Nearest neighbor algorithms, in general, do not reflect the order in which nodes are added. Sibson (1973) showed how algorithms can be modified to retain the order. Basically, with Prim's algorithm a secondary matrix is kept to track the order that the documents are selected. Sibson further showed that a dendrogram can be readily constructed from such an order matrix (Rohlt, 1982; Sibson, 1973). The order matrix is typically referred to as a pointer representation of a dendrogram. The pointer representation of the example is

order	1	2	5	4	3
distance	.1	.3	.4	.2	NA

Document 1 was the first document added, and its nearest neighbor is document 2 at a dissimilarity of .1, document 2's nearest neighbor is document 5 at a dissimilarity of .3, etc. The research undertaken provides the pointer representation as the final product. This is in keeping with the current literature on parallel classifications (Dahlhaus, 1992; Olson, 1995). The dendrogram of a large document set is very cumbersome, its actual presentation is left for continued research in the areas of information visualization (AIPASG, 1995; Kaufman, 1994; Risch, May, Thomas, & Dowson, 1996).

Results

As in the clustering case, the algorithm was run on a 5000 document sets and a 10000 document set. Document vectors are packed together into a single large file. This file is read by the various nodes and the individual documents are broken out within the program. This breakout is based on a document's offset from the start of a block. Each node was given responsibility for a certain number of documents in the algorithm. The number of documents per node is basically D/P, with D the number of documents and P the number nodes.

As previously stated, there is no need to keep large amounts of data during the execution of the algorithm. For this reason, the block size was increased from a block size of 64K to a block size of 128K. Larger block sizes of 256K and 512K were also used.

Run Data. We ran our results on a production system. The system time queues prohibited runs that exceeded 12 h of execution time. These time constraints did not allow runs on the 10,000 document set to be completed for a configuration of under four nodes. Results are only provided for 4, 8, and 16 node systems. Speed up for these systems are adjusted to account for the 10,000 document system beginning at four nodes. The optimum speed up becomes based on a factor of P/4. For example, the linear speed up of a system going from four to eight nodes would be for the eight node system time to be $\frac{1}{2}$ the time of the four node system. Therefore, the "linear" time for speed up at eight nodes would be 2. At 16 nodes, the "linear" speed up would be 4. Results are shown in Table 4, as well as Figure 6.

Speed up drops considerably as processors are added. There are several reasons for this. First, is the effect of message passing. Each node must know which document is selected as the next vertex. This requires the transmission of a message. As the number of nodes increases, the overhead of message passing increases. As the number of nodes increases, the number of documents each node is responsible for decreases. This means there is less work being done between messages. These are typical side effects of parallelism in an MIMD architecture.

Another factor to consider is documents that become inactive. When a document is added to the tree, it is no longer compared to any other document. Not only does the number of documents per node decrease as a factor of nodes, but the number of active documents decreases as the algorithm progresses. So the amount of work between messages is also decreasing as a natural process of the algorithm's execution.

TABLE 4. Times and speed up for (a) 5000 documents and (b) 10,000 documents.

	128K		256K		512K	
P	Time	Speed up	Time	Speed up	Time	Speed up
a		1		1		1
1	20883.14		20398.36		20181.06	
2	11541.20	1.81	10794.73	1.89	10628.15	1.90
4	6795.09	3.07	5925.83	3.44	6135.10	3.29
8	4295.68	4.86	3523.24	5.79	4071.97	4.96
b		1		1		1
4	23310.74		22091.93		21226.19	
8	15062.78	1.55 (2)	13370.43	1.65 (2)	12375.06	1.72 (2)
16	11030.62	2.11 (4)	8883.09	2.49 (4)	9415.6	2.25 (4)

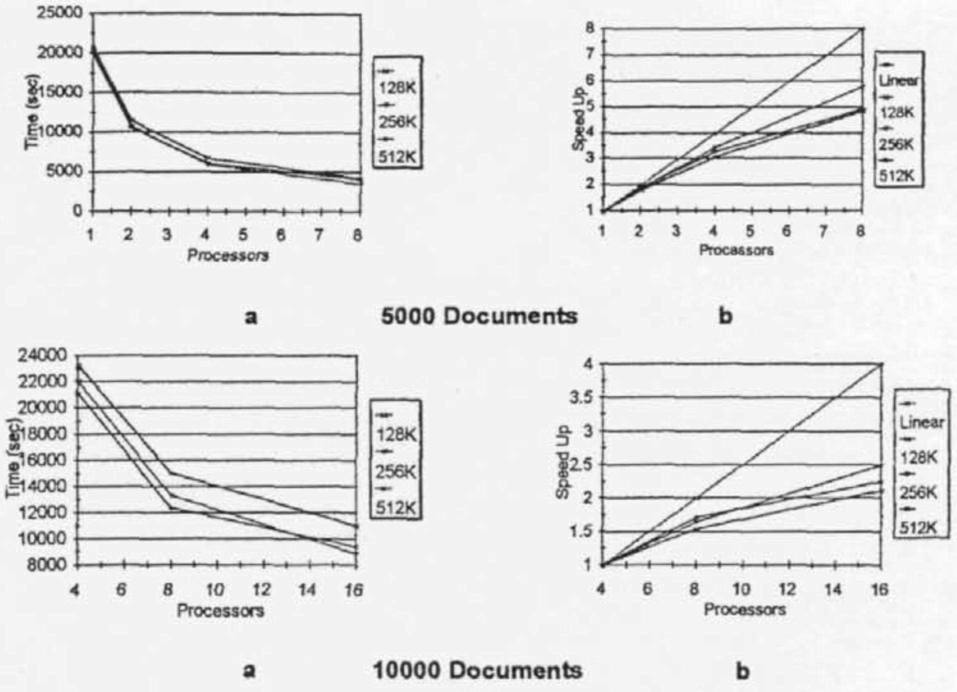

FIG. 6. Time (a) and speed up (b) for 5000 and 10,000 document sets. Block sizes of 128K, 256K, and 512K are superimposed.

As anticipated, the 512K block size provided improvements over the 128K block size. But performance decreased with respect to the 256K block as processors were added. The decreased performance is not intuitive. An explanation is presented in the next section.

The Overlap Factor, an Indicator of Effects of File I/O. In this algorithm the document vectors are packed into a single file. If the program is reading blocks of size 128K, then the data file, and corresponding offset file, must be based on 128K blocks. If the program is reading blocks of size 512K, the data file and corresponding offset file are based on 512K, etc. Regardless of the block size used, the individual document vectors do not change in size nor in number. The vectors are in the same order within the data file whether the file is built in 128, 256, or 512K blocks. The fact that the data does not change leads to the belief that larger blocks should be better. That is, intuitively, a given set of data built in 128K blocks, should be read faster in 256K blocks, and faster still in 512K blocks.

However, as the block boundaries are shifted, the data contained within those blocks changes. For example, a document vector may be in block 21 when based on 128K blocks, block 11 using 256K blocks and block 6 using 512K blocks. When the algorithm is run, it apportions documents among the processors. Note that it is the document vectors that get apportioned and not the data blocks. Therefore, it is possible that a data block may be shared by two processors. Fortunately, it is possible to analytically and a priorily determine exactly which block contains which document vectors. Based on the number of processors, it is possible to determine how many blocks are shared by the processors. The number of blocks being used by each processor can be summed together to give the total number of blocks being read by the program. The Overlap Factor, which is a measure of this overlap of blocks across processors, has an optimal value of P,

TABLE 5. Overlap factor, OF, for (a) 5000 and (b) 10,000 document sets.

	a			b		
P	128K	256K	512K	128K	256K	512K
1	1	1	1	1	1	1
2	2.05	2.09	2.18	2.02	2.05	2.09
4	4.29	4.57	5.46	4.14	4.28	4.57
8	11.24	10.67	12.36	8.67	9.33	10.67
16				18.89	21.71	27.43

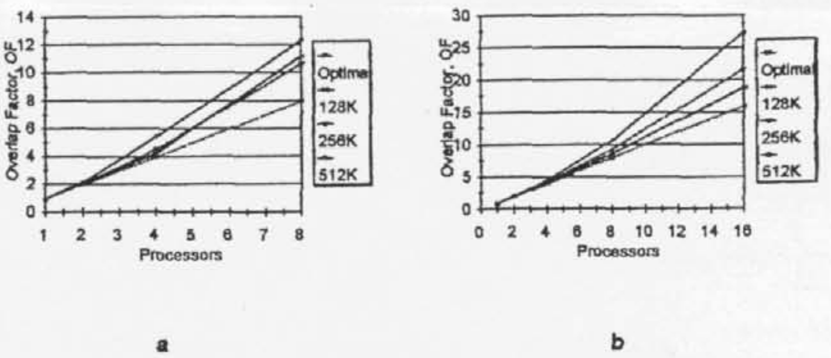

FIG. 7. Overlap factor (a) 5000 and (b) 10,000 document set.

the number of processors. The Overlap Factor has been computed for the 5000 and 10,000 document sets for each block size as shown in Table 5.

The overlap factor is an indicator of the effects of I/O which is imbedded in an algorithm's execution. With this in mind, in Figure 7(a) we show the significant impact due to I/O at eight nodes. With an optimum value of *P*, it appears the least effected by I/O will be the 256K block. This is supported by the speed up curves for 5000 documents (see Fig. 6).

As seen in Figure 6, the speed up curves for the 10,000 document set also begin flattening out at eight nodes. As explained before, this is because the systems with more nodes are actually reading in more data. The overlap factor curves of Figure 7(b) indicate a significant impact of I/O from 8 to 16 nodes. It must be reiterated that the overlap factor is just an indicator of the effects of imbedded I/O in an algorithm. As the data block is read in, the algorithm separates data from the block. This is typically a faster operation than block reading. The total requirement (e.g., block reading and data dispersion) is the determining factor in performance. In the case of the 10,000 document set, the lower overlap factor of the 128K block size affects but does not fully compensate for the amount of data the 256K can disperse between block reads, hence the 256K block size has the best overall performance.

Conclusion

Large document bases are becoming more common place. The prohibitive execution time of traditional processes operating on large document sets has been a major concern in cluster-based information retrieval approaches (Salton, 1989). We focused on the single-pass clustering algorithm and the single-link classification algorithm, each a popular approach to clustering and classification, respectively. We used parallel computing to show that these operations, once considered computationally prohibitive and, hence, infeasible, are indeed capable of meeting the needs of the community in dealing with large, dynamic document bases.

In some circumstances, the clustering process was reduced from approximately 6 h for 5000 documents in a sequential system to under 30 min in a multiprocessor environment. Similar results were attained with the 10,000 document set, with clustering time reduced from over 13 h to approximately 1 h.

Document classification has relied on having ancillary matrices of data readily available. As seen, these ancillary matrices may themselves, require significant computing resources and large amounts of storage. We detailed a process which removes the requirement for these ancillary matrices by utilizing the power of multiple processing systems to compute them as needed. Even with this additional computational burden, the classification of the 5000 document set was reduced from almost 6 h to approximately 1 h on an eight node system.

The results of our experiments show a step forward in meeting the requirements to classify, group and process large document sets within nonprohibitive execution times. This has been demonstrated on a currently commercially available, multiprocessor computer; the Intel Paragon, operating in a production environment.

Acknowledgments

This work was supported in part by a grant from the DoD Major Shared Resource Center operated by the Aeronautical Systems Division, at Wright-Patterson Air Force Base and the National Science Foundation under contract # IRI-9357785.

REFERENCES

Aho, A. V., Hopcraft, J. E., & Ullman, J. D. (1983). *Data structures and algorithms.* Reading, MA: Addison-Wesley Publishing Company.

AIPASG. (1995). *P1000 strategic plan for information visualization,* version 1.6. Washington, DC: IC Advanced Information Processing and Analysis Steering Group.

Anderberg, M. R. (1973). *Cluster analysis for applications.* New York: Academic Press.

Cutting, D. R., Karger, D. R., & Pedersen, J. O. (1993). Constant interaction-time scatter/gather browsing of very large document collections. *ACM SIGIR '93*, 126–134.

Dahlhaus, E. (1992). Fast parallel algorithm for the single link heuristics of hierarchical clustering. *Proceedings of the Fourth IEEE Symposium on Parallel and Distributed Processing*, (pp. 184–187).

del Rosario, J. M., & Choudhary, A. N. (1994). High-performance I/O for massively parallel computers. *IEEE Computer*, 59–68.

Jain, A. K., & Dubes, R. C. (1988). *Algorithms for clustering data.* Englewood Cliffs, NJ: Prentice Hall.

Kaufman, A. E. (1994). Guest editor's introduction: Visualization. *Computer, 27*, 18–19.

Kumar, V., Grama, A., Gupta, A., & Karypis, G. (1994). *Introduction to parallel computing, design and analysis of algorithms.* Redwood City, CA: Benjamin/Cummings Publishing Company Inc.

Kwok, K. (1990). Experiments with a component theory of probabilistic information retrieval based on single terms as document components. *ACM Transactions on Office Information Systems*, 363–367.

Li, X. (1990). Parallel algorithms for hierarchical clustering and cluster validation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 12*, 1088–1092.

Nastea, S., El-Ghazawi, T., Frieder, O. (1996). A statistically-based multi-algorithmic approach for load balancing sparse matrix applications. *Proceedings of the IEEE Symposium of Frontiers of Massively Parallel Computation, Annapolis, MD* (pp. 180–187).

Olson, C. (1995). Parallel algorithms for hierarchical clustering. *Parallel Computing, 21*, 1331–1325.

Pogue, C. A., Rasmussen, E. M., & Willet, P. (1988). Searching and clustering of databases using the ICL distributed array processor. *Parallel Computing 8*, 399–407.

Quinn, M. J. (1994). *Parallel computing theory and practice.* New York: McGraw-Hill, Inc.

Rasmussen, E. (1991). Introduction: Parallel processing and information retrieval. *Information Processing & Management, 27*, 255–263.

Rasmussen, E., & Willet, P. (1989). Efficiency of hierarchic agglomerative clustering using the ICL distributed array processor. *Journal of Documentation, 45*, 1–24.

Reddaway, S. F. (1991). High speed text retrieval from large databases on a massively parallel processor. *Information Processing & Management, 27*, 311–316.

Risch, J., May, R., Thomas, J., & Dowson, S. (1996). Interactive information visualization for exploratory intelligence data analysis. *1996 IEEE Virtual Reality Annual International Symposium (VRAIS '96).*

Rohlf, F. J. (1982). Single link clustering algorithms. In P. R. Krishnaiah, J. N. Kanal, Eds. *Classification, pattern recognition, and reduction of dimensionality.* (pp. 267–284). (Handbook of Statistics, Vol. 2), Amsterdam: North Holland.

Romesburg, H. C. (1984). *Cluster analysis for researchers.* Belmont, CA: Lifetime Learning Publications.

Salton, G. (1989). *Automatic text processing: The transformation, analysis, and retrieval of information by computer.* Reading, MA: Addison-Wesley Publishing Company.

Sibson, R. (1973). SLINK: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal, 16*, 30–34.

Spath, H. (1982). *Cluster analysis algorithms.* New York: John Wiley and Sons.

Willet, P. (1988). Recent trends in hierarchic document clustering: A critical review. *Information Processing and Management, 24*, 577–597.