TR-IT-0155

Massively Parallel Document Retrieval in Clustered Databases

Detlef Koll Eiichiro Sumita Hitoshi Iida

March 5, 1996

Abstract

Our goal has been to develop an effective and efficient document retrieval system for very big databases, based on the vector space model. Thus we (1) implemented a massively parallel retrieval kernel on a SIMD-machine and (2) devised a fast non-hierarchical clustering method for restricting its search scope without hurting retrieval effectiveness.

This paper discusses score-computing algorithm and load-balancing method of the parallel kernel, the document clustering method and how those two parts combine to a large-scale retrieval system.

Evidence for the efficiency and effectiveness of this approach is given for standard test suites: (i) Virginia-collections; (ii) Tipster-collection with some gigabyte of text.

Contents

Chapter 1

Introduction

1.1 Introduction

In the past decades the Vector Space Model (VSM) proved to be a effective and easy-to-use method of document retrieval. The Usability of a retrieval system can be enhanced further by relevance feedback techniques (e.g. [SaBu90]), which reformulate an initial, manually generated query in several iterations according to relevance judgements of the user for formerly presented documents.

However, the computational requirements for processing those automatically generated queries increase drastically. Re-formulated queries tend to contain considerably more terms than the initial query, depending on the size and number of documents, that were used for relevance feedback, and within those typically a big proportion of the high-frequency terms of the database.

Let f_w denote the frequency of occurrence of word w in the document collection, i.e. the number of documents in which the word w occurs. Then including word w in the query changes the evaluation of f_w documents by matching the occurrence of w in those documents. The number of terms in the document collection that are matched by a query $\mathcal{Q}$ is expressed by

$$M_{\mathcal{Q}} := \sum_{w \in \mathcal{Q}} f_w$$

If we set this value in relation to the overall number of words in the document collection (i.e. the sum of the f_w, where w ranges over all distinct index), we get the percentage of database-terms matched by a query. This percentage depends heavily on the number and size of relevant documents that were used to re-formulate the initial query: Table 1 demonstrates this for several standard test sets: the original queries of the investigated test

Database	Original Queries	Queries re-formulated with # relevant documents		
		5	10	30
CISI	4.9%	25.0%	32.6%	48.7%
Cranfield	1.5%	24.8%	34.2%	60.3%
LISA	3.2%	17.1%	27.4%	40.8%
Tipster Vol 1	2.2%	15.3%	30.4%	50.3%

Table 1.1: Average percentage of database terms matched by original and re-formulated queries (IDE-DEC-HI-formula [SaBu90])

collections match on average between 1% and 5% of the document vector terms. A query re-formulated with 30 relevant documents already approximately 50% — independent of the size of the database in concern — making the retrieval process a computationally very demanding task.

Conventional ways to overcome this problem:

I. By using an inverted-index-representation of the database only those terms of the database have to be processed, that are actually matched by a given query. This method allows to skip all document terms, that do not have any effect on the outcome of the search, because they correspond to words, that do not occur in the query. Thus the outcome of the VSM-search process is not changed by using an inverted index.

II. Restrict the search scope to the most promising parts of the document collection by clustering the document database and examining only those clusters, that are for a given query likely to contain relevant documents.

Document clustering methods have been used for this purpose for long: Early work concentrated on non-hierarchical document clustering methods [Salt71] with the objective to increase the efficiency of the retrieval process. Due to their lack of theoretical soundness and their minor effectiveness they lost however soon interest and were all but abandoned in the last years. Hierarchical document clustering methods took over their role and showed good results, both in terms of retrieval effectiveness and efficiency [Voor86] [RiCr75] for a variety of mostly relatively small databases. The cost of clustering document collections is however for most of them (containing the more effective methods like complete link and HBC [IwTo95]) prohibitively high for large collections.

So in spite of being thoroughly investigated on small test-databases, clustering mechanisms are very seldomly applied to real world collections for retrieval purposes. (The work of [CKP92] [CKP93] used clustering mechanisms on big databases in order to support document browsing.)

III. The intrinsic parallelism of the computation intensive part of the retrieval process led to a number of parallel implementations for various parallel architectures and retrieval models.

To mention but a few: [StCa88] implemented an overlap-encoded signature based system on a connection machine CM-2, a SIMD[1]-machine with up to 65536 processing elements. The document representation can be stored to a parallel disk array (PDA) and thus the resulting system is able to handle very big databases. But although retrieval speed gains were obtained, the level of retrieval effectiveness was not satisfactory due to the use of binary (not weighted) vector representations.

Parallel implementations of the pure vector space model have been reported for example by [EGMS95], using a Parsytec GCel3 with 512 processing nodes (MIMD) and by [ASI95a] using a KSR-2 with 32 processors (MIMD). Both obtained considerable performance gains with almost linear speedup.

[1]SIMD: single instruction multiple data,
MIMD: multiple instruction multiple data (classification method for parallel architectures due to Flynn)

Our work has been aimed at building an retrieval system, that is efficient and effective for very big databases.

For its effectiveness the well known vector space model has been chosen to base the retrieval system.

In order to meet the claim of efficiency, a massively parallel SIMD-machine (MasPar MP-2, 8192 processing elements) has been used as hardware platform for the retrieval kernel.

We explicitly do *not* assume, that it is possible to store the whole document collection in the RAM of the underlying machine, not even in the internal representation which is used for the retrieval process.

For processing a user posed query it is thus inevitable to load the database or at least parts of it from a secondary storage medium. The VSM-retrieval phase has the same space and time complexity for processing a single query[2] and require only very few instructions per loaded term of the database. The time required to transfer the database from secondary to primary storage will thus dominate the computing time.

So it is necessary to review also methods of reducing the percentage of the database that is examined for an individual query (I,II), in order to avoid the bottleneck of transferring the whole database to and fro a secondary storage for each individual query.

Since the benefits of using an inverted index in reducing the computational and IO-requirements of the retrieval phase is clearly bounded by the percentage of database-terms that is actually matched by a given query (cf. Table 1), the use of an inverted index (I) will not pay in the assumed relevance feedback environment.

The use of clustering methods has however the potential of reducing the search scope of the VSM to a manageable fraction of the database. For reasons that we will discuss in chapter 2.2.1 we fell back to a non-hierarchic document clustering approach, in spite of being considered in recent research to be less effective than hierarchical ones.

Consequently this work comprises the following topics:

1. development of a massively parallel retrieval kernel on a MP-2
2. design and investigation of clustering methods for big document collections
3. combination of the retrieval kernel and clustering methods to a document-retrieval system

The rest of this chapter gives a brief introduction in the vector space model and the architecture of the MasPar MP-2. The massively parallel document retrieval system for clustered databases resulting from this work, the CRISP[3], will be described in chapter 2. The discussion of the retrieval performance of the CRISP-system will form the content of chapter 3 and 4.

1.2 The Vector Space Model

The fundamental problem of document retrieval is, given a collection of documents, how to identify efficiently and effectively the documents susceptible of satisfying an imperfectly described user need.

[2]Things are different for batch mode query-processing, where several queries are evaluated at the same time.

[3]Cluster-based RetrIeval with Simd-Parallel machines

In the Vector Space Model (VSM) proposed by Salton 1971 [Salt71] documents and queries are represented by weighted vectors in the n-dimensional space, where n denotes the number of distinct index terms (stems of words that were used for retrieval purposes) of the document collection. During query processing, documents are ranked according to their similarity to the query vector, usually measured by the cosine angle between query and document vectors and returned to the calling instance according to this ranking.

A VSM based retrieval systems consists of at least the following four components: the plain text database, a vector generator, a database of document vectors and a unit for processing queries.

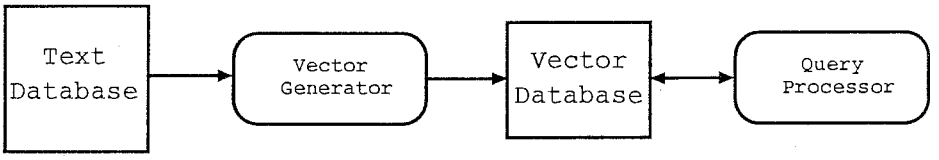

Figure 1.1: Block diagram of a VSM based document retrieval system

The database contains the texts of the documents or articles to be searched. The generator operates directly on the text database to produce a usually much smaller database of weighted vectors, which is used by the query processor to rank the documents of the collection according to their similarities to a given query.

1.2.1 Database Representation

The generation of a weighted vector is carried out by running the following three preprocessing operations:

- The stop list filter removes the most frequently occuring words (such as *and, of, or, but, the, etc, ...*) from the text of each document or query. These words are poor discriminators, and their removal has no effect on the retrieval effectiveness. Moreover, the filtering process reduces storage requirements and increases query processing speed. The filter was applied using a stop list consisting of 425 words derived from the Brown Corpus [FrKu82].

- The suffix stripping stemmer replaces the words preserved by the stop list filter to their stem forms. For example, the stemmer replaces a variety of different forms such as *analysis, analyzing, analyzes,* and *analyzed* by a common word stem *analy.* The stemming operation reduces storage requirements since many words are replaced by a single stem word. Furthermore, it might increase the retrieval effectiveness since the stem word has a higher frequency of occurrence than that of the words replaced. In this system we used the well-known Porter stemming algorithm [Port80].

- The weighter assigns a real-number weight to each word stem produced by the stemmer. The weight distinguishes the degree of importance of the word in the document (query), and thus leads to improved retrieval effectiveness. Moreover, it adds user-friendliness to the system as it facilitates ranking of the retrieved documents. In this system, we used the following weight assignment function for both the documents

and the queries [SaBu88].

$$w_i = \frac{(0.5 + 0.5\frac{f_i}{\max f}).log\frac{N}{n_i}}{\sqrt{\sum_{j=1}^{j=W}((0.5 + 0.5\frac{f_j}{\max f})^2.(log\frac{N}{n_j})^2)}}$$

where,
w_i: weight of word i in the document (query).
f_i: frequency of occurrence of word i in the document (query).
n_i: number of documents(queries) to which word i is attached.
N: number of documents (queries) in the database.
W: total number of words in the document (query).

The denominator of the function above is a weight normalization component which ensures that the lengths of document (query) vectors are equal. The function assigns weights varying between 0 and 1, where 0 represents a word that is absent from the vector. The resulting document vectors are consequently very sparsely filled.

1.2.2 Query Processing

Taking as input the vector representation of a user-formulated query, the query processor computes a ranking of the document collection according to the similarities between query and document vectors.

The common measures of similarity between queries and documents are based on the inner product between their corresponding vector representations. Those include for example the Dice-, Jaccard- and Cosine-coefficient M_c,

$$M_c(q,d) = \frac{\sum_{i=1}^{n} q_i d_i}{\sqrt{\sum q_i^2 * \sum d_i^2}}$$

the latter of which we used in the current implementation. The computationally demanding, i.e. the for a parallel implementation interesting, part of the VSM based search process is formed by the computation of the document-query interrelation.

The top-ranked n documents are returned to a user, who will judge the documents for their relevance to the example query and, in case his information need is not satisfied, will return the relevance judgements to the query processor.

The query vector is reformulated by expanding and re-weighting its elements according to the following Ide dec-hi relevance feedback method [Ide71]:

$$Q^{new} = Q^{old} + \sum All\ \ rel.docs - Top\ \ nonrel.doc$$

Q^{new} is the new query vector which is obtained by (1) adding to the previous query vector Q^{old} all words and corresponding weights of all relevant documents vectors, and (2) subtracting from the new query vector all words and corresponding weights found in the top ranked non-relevant document vector. Query reformulation using the Ide dec-hi method has proved to be superior to many other relevance feedback methods [SaBu90].

1.2.3 Remarks

Some remarks:

- We will not always make a clear distinction between 'documents' and 'document vectors', respectively 'queries' and 'query vectors'. As well we will – a little bit inaccurately – use 'word' as a synonym for 'index term', describing the units of document and query vectors.

  Since this report is only concerned with the query processing part of a VSM-retrieval system this should not lead to confusions.

- The VSM described above is actually only one variant of VSM based document retrieval methods. The CRISP-system extends however easily to a number of related methods:

  - similarity function:
    the CRISP system is designed to compute similarity functions between one input vector (query) and a given set of document vectors, based on the inner product between query and document vectors.

    As mentioned above this class comprises the most common similarity functions, as for example the cosine- Dice- and Jaccard coefficient.

  - relevance feedback methods:
    the ranking routines of the CRISP should enable it to be applicable for a wide class of relevance feedback methods, including for example the Rocchio-methods, or the Common Term System [ASI96]. For restrictions confer to the discussion of the document ranking routine, 'Document Ranking' in 2.1.2.

1.3 The MasPar MP-2

The MasPar-2208 SIMD-computer used in this work consist of the following main components (cf. Figure 1.2):

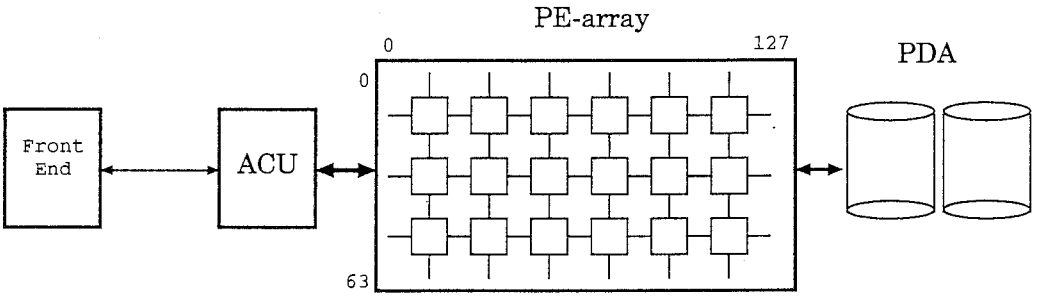

Figure 1.2: Structure of a MasPar MP-2208.

- a Front-End-Workstation
- the array control unit (ACU)
- an array of processing elements consisting of 8192 processing elements (PEs) arranged in a two-dimensional mesh of 128 × 64 processors. Processors are connected

in this mesh to their four neighboring PEs (north, south, east, west) by a high speed local communication structure. Each PE is equipped with 64 K-Bytes of local RAM, making a total of 512 M-Bytes local memory capacity for the PE-array.

- a parallel disk array (PDA) consisting of 8 hard-disks (in total 11.5 gigabyte).

A MasPar-program consists of two parts: (1) a sequential part, executed on the front-end and (2) a parallel part that is executed on ACU and PE-array of the MasPar.

The sequential part consists mostly of routines for interacting with a user-interface (usually running on a second, external workstation), for creating parallel data-structures and for communicating with and calling of the parallel part of the program.

SIMD-parallel machines execute, as sequential machines, only one operation at a time. The array control unit decodes the program instruction and executes the part of the SIMD-code, that operates on singular data. Instructions operating on plural data are translated in a sequence of micro-code instructions and broadcasted for execution to the PE-array. Processing elements can either participate in the current computations on their own local data, or pause until the other processors executed the current instruction.

The peak performance of a MP-2208 is reported to lay somewhere around 3 G-Flops using 8192 processing elements .

Chapter 2

The CRISP-system

Figure 2.1 gives an overview over the structure of the CRISP-system. CRISP consists of two major components:

1. the *Parallel Retrieval Kernel*,
   an implementation of VSM-based document retrieval on a MasPar MP-2,

2. the *Cluster-Subsystem*,
   routines for clustering document collections and for searching the resulting clustered databases.

Query processing of the CRISP system is as follows: a user formulated query is translated as ususal into its vector representation and possibly re-formulated by a relevance feedback process. The resulting query vector forms the input of the CRISP retrieval system. The query processing divides into three steps:

i. rank the cluster centroids according to the query

ii. select and load the top-ranked $r\%$ clusters

iii. rank the documents of those clusters and return the locations of the top-ranked n documents

In contrast to the regular VSM, CRISP's search process does not produce a ranking of all documents of the collection, but only of a part of it, whose size r depends on the users need of precision and response time for an individual query. In order to detect the parts of the document collection that most likely contain relevant data, an incoming query is first used to rank the centroids of the clustered document collection, in a second step to compute the scores and ranking of the documents, that are contained in the top-ranked clusters. The highest ranked documents are returned to the calling instance.

Since the parallel implementation of the VSM is largely independent of the cluster-search routines of the kernel, we will discuss it isolated from the rest of the retrieval system in 2.1 and describe the clustering and cluster-search methods in chapter 2.2.

2.1 The parallel retrieval kernel

Figure 2.2 shows a flowchart of the retrieval part of CRISP without clustering routines and cluster-search mechanisms.

The database and query pre-processing (filtering, stemming, weighting) is the same for the parallel retrieval kernel of CRISP and for the VSM-implementation on the KSR

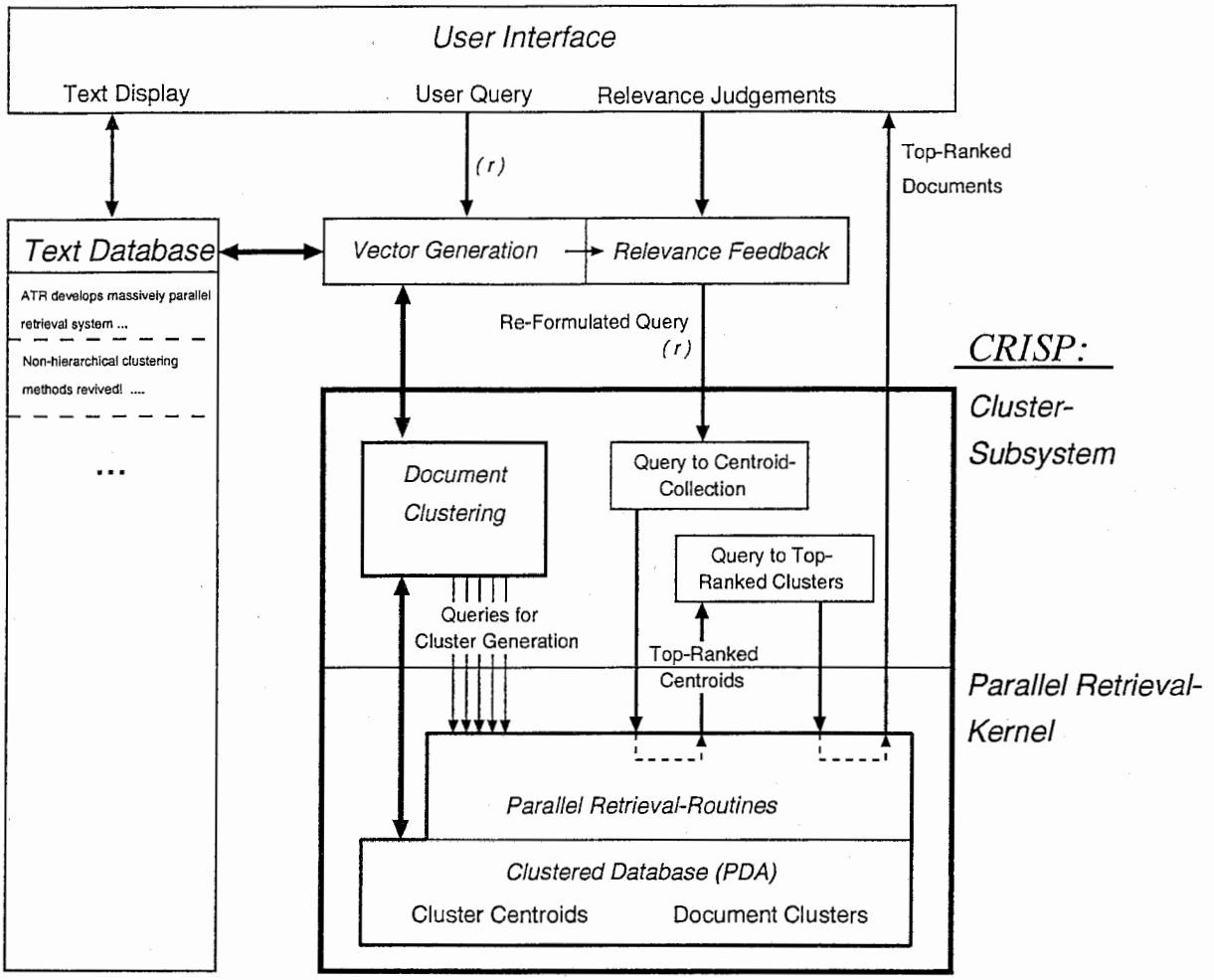

Figure 2.1: Structure of the CRISP-system

[ASI95b]. But in contrast to the KSR-implementation, the Maspar-kernel needs some more pre-processing steps, in which a distributed representation of the document vector collection is computed and stored to the parallel disk array of the MP-2.

The vector representation of the query is generated analogously, the distributed data structures for the query vector are build according to the distribution of the database (cf. 2.1.1) and are broadcasted to the PE-array.

The storage capacity of the MP-2 is by assumption not big enough to store the vector representation of the complete document collection. Thus for a given query the document collection or at least some selected parts of it (cf. 2.2.1) have to be loaded from the parallel disk array. The scores of the respective document vectors are computed (*Parallel Inner Product*) and ranked (Incremental Parallel Ranking).

If the part of the database that has been selected as the search scope is bigger than the local memory capacity of the PE-array, the loading, scoring and ranking steps are iterated until the full search scope has been examined. The document ranking is build incrementally over the iterations by ranking the scores of the documents in each iteration and successively merging the current local ranking with the result of prior iterations.

The resulting list of the top-ranked documents is returned to the calling instance for relevance feedback.

The performance of a SIMD system depends heavily

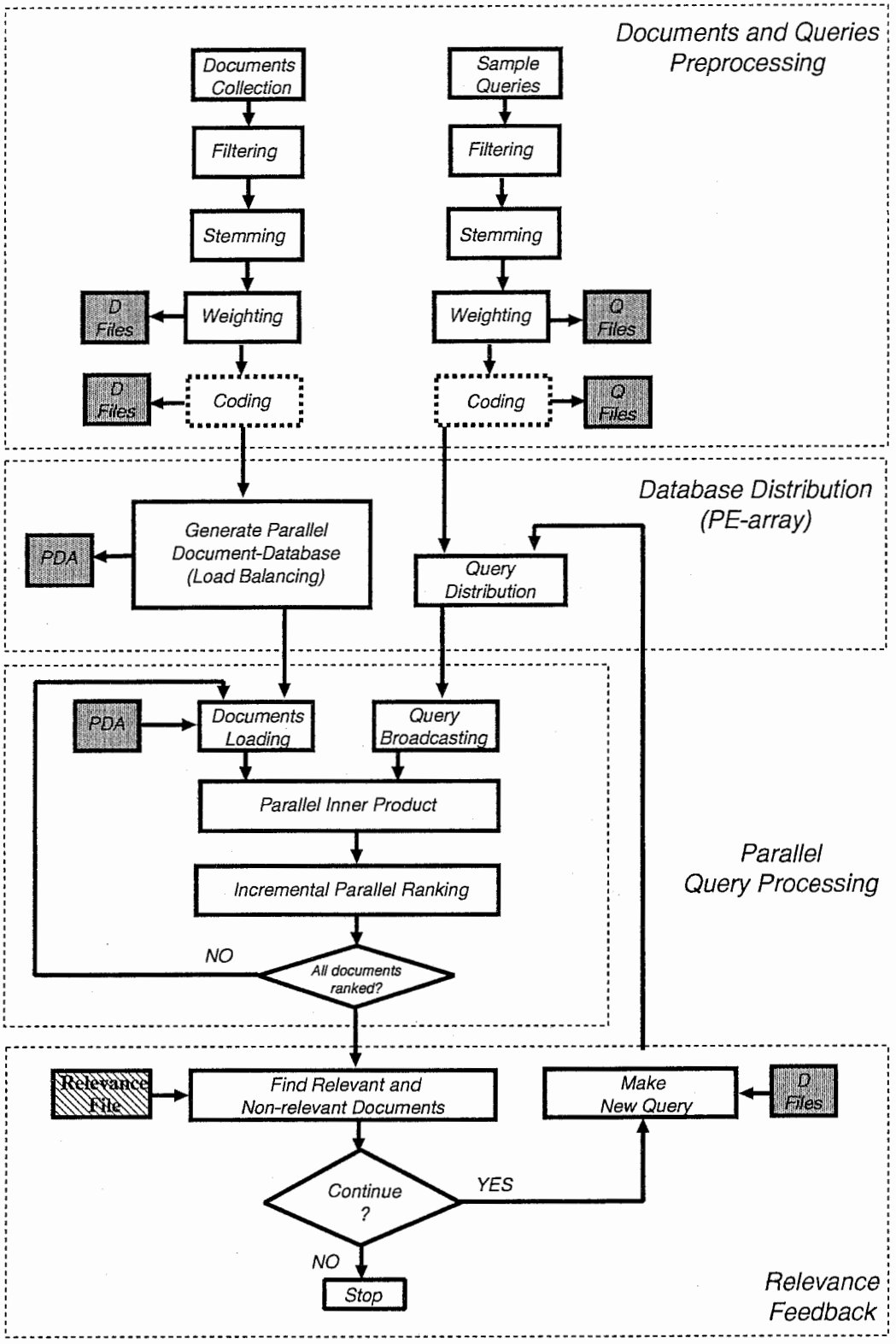

Figure 2.2: Implementation flowchart of the MP-2 retrieval system

(1) on how evenly the computational load is balanced between the processing elements: the processing element with maximum load determines the computing time of the full system.

(2) on the homogeneity of program instructions performed on the local data of the processing elements.

Thus an efficient SIMD-implementation has to assure, that no processing element gets significantly more work to do than its colleagues and that all processing elements want to execute the same instructions on their share of data at the same time.

The first point (1) holds for almost all parallel machines, independent whether it is a SIMD or MIMD-architecture. But while the distribution of the document vectors on the processing elements is a minor problem on systems with relatively few processing elements (due to the high inherent parallelism of the retrieval phase) and high local memory capacity, it becomes increasingly critical with growing number of processing elements, respectively with decreasing local memory capacity. The load balancing routines required for generating the evenly distributed database are described in 2.1.1.

The second point (2) is specific to the SIMD execution model. Section 2.1.2 describes the SIMD-parallel algorithms for scoring the document collection and ranking it according to this score.

The selection and loading of the parts of the database, that have to be examined for a given query are the task of the cluster-search routines. Its discussion will be postponed to chapter 2.2.1.

2.1.1 Load balancing routines

Some considerations concerning the load distribution of the query processing-task on a MP-2 (This list of 6 straightforward arguments contains some basic design descisions of the CRISP. They are not a cogent string of arguments, rather they describe only one possible solution for the given parallelization problem.)

i. The scores of many documents are computed at the same time for one and the same query, resulting in multiple concurrent accesses to different parts of the corresponding query vector. To serve this requirements, the *query-vector* should be *stored in the local memory* of the PEs.

ii. Queries that were re-formulated in a relevance feedback process might well contain some thousand terms with non-zero weight. Thus it is impracticable to store a complete copy of the query-vector on each processing element. Instead, one has to *split the query vector and distribute the parts on the local memory of a set of PEs.*

iii. The distribution of the query vector determines where the terms of document vectors should be located on the PE-array: corresponding terms have to be stored to the same PE. Thus the *terms of each document are distributed on a set of PEs.*

iv. In order to reduce the communication overhead caused by recombining the partial results, *each document has to be stored on a small number of adjacent processing elements.*

v. *Each PE contains parts of more than one document* in order to exhaust the PE-arrays storage capacity.

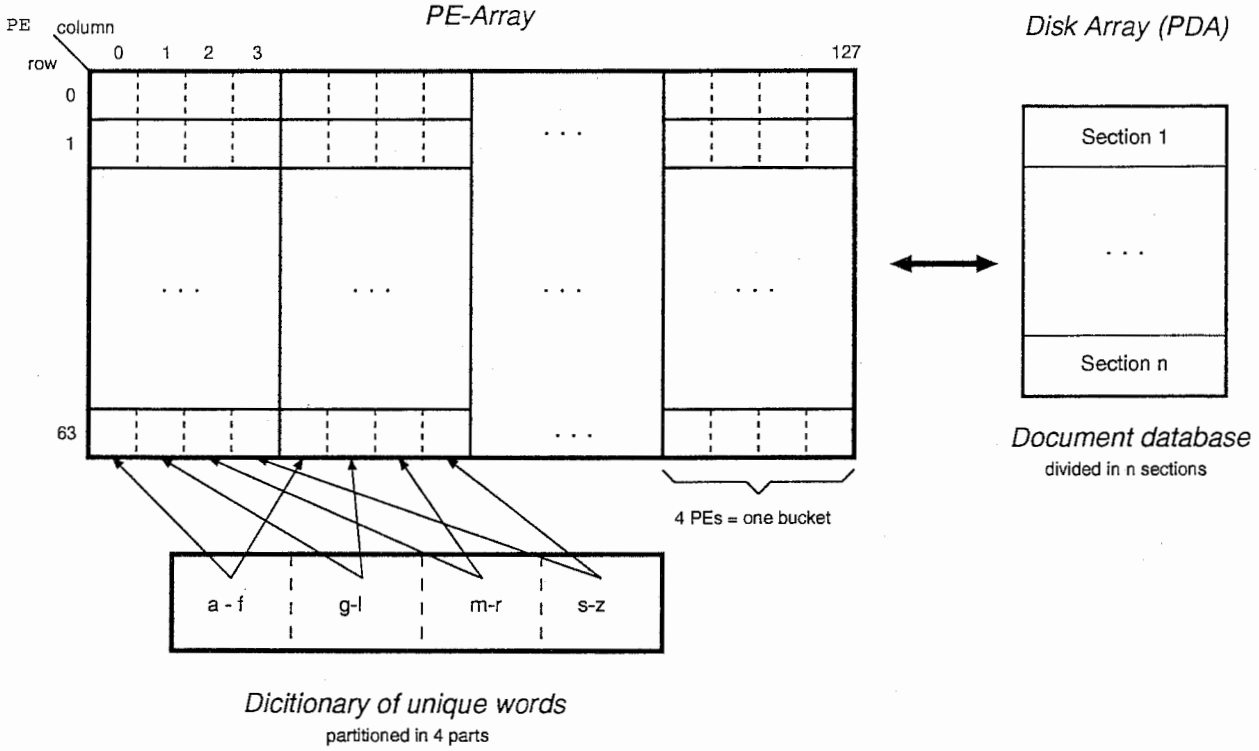

Figure 2.3: The PE-array is subdivided into buckets. Each processing element of a bucket is assigned to one part of the partitioned dictionary.

vi. Finally, the storage capacity of the MasPar MP-2 may not be sufficient to store the whole database. In this case we have to *divide the database into several sections* and process a sufficiently small subset of those parts at a time.

The distribution of document and query vectors, claimed in points *i.* and *ii.*, can be defined uniquely by partitioning the dictionary of distinct words of the document database and dedicating each PE to one dictionary partition (2.1.1). PEs will only store and compute terms of document- and query-vectors that correspond to the words of the partition to which they are assigned. In the following we call a row of adjacent PEs, which covers the whole dictionary exactly once, a *'bucket'*.

Figure 2.3 gives an overview of the resulting PE-array organization of the database. Figure 2.4 describes the data structure of one bucket, containing a copy of the distributed query and 3 document vectors, in more detail.

Let us assume for the moment, that the distribution of documents on the sections of a database *vi.* is known (we will discuss this in 2.2.1, using the term 'clusters' instead of 'sections'). Given the distribution of documents on the database sections and the distribution method for assigning the document terms to the processing elements of a bucket, it remains to be addressed, which document to place on which bucket of the PE-array. This is of some importance, since this distribution determines the maximum number of terms per PE (in Figure 2.4 on PE 126) and therewith not only the computation time, but also the storage requirements of the parallel database (storage overhead caused by the parallelization). This part of the load balancing routines is discussed in 2.1.1.

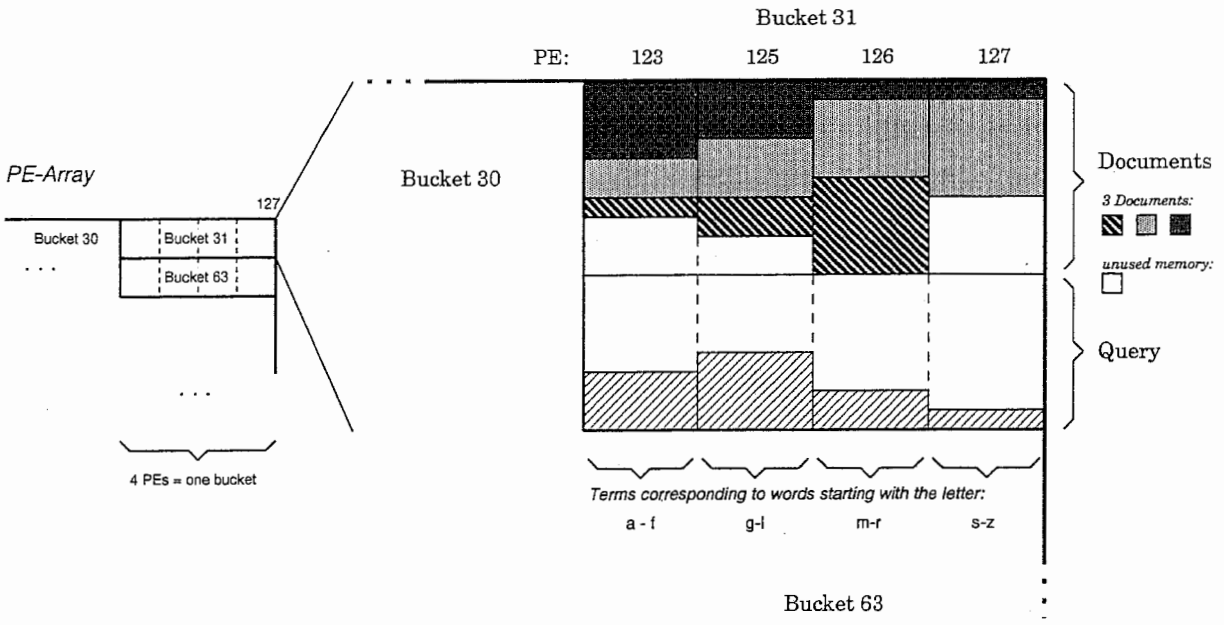

Figure 2.4: Each bucket contains the terms of several documents and a copy of the query, distributed according to the partitioning of the database.

Dictionary Partitioning

The distribution of document and query terms on the PEs of a bucket is determined by partitioning the database of distinct words and assigning each PE to exactly one partition of the dictionary. PEs will only store and compute those terms of the document collection, that correspond to words of their dictionary part.

In the example of Figure 2.4 and 2.1.1 the dictionary has been divided into 4 disjunct sets. Consequently, each document and query-vector is divided into four parts, each containing the terms of only one partition. The 4 parts of each vector are stored on a row of four adjacent processing element (a 'bucket').

Thus the term distribution is fully described

1. by chosing a number n of dictionary partitions and
2. by assigning each word of the dictionary to one partition.

Since the queries that will be posed to the retrieval system are not known during the generation of the distributed database, one cannot guarantee the distribution of query terms on the processing elements of a bucket to be even. Assuming that each term occurs with the same probability in a query, and that this probability is independent of the other terms in the query[1], the best one can do to assure an even distribution of query terms is to assign the same number of words to each dictionary partition .

Let the number of dictionary partitions be denoted by n and the overall number of words in the dictionary by $|\mathcal{D}|$. Then each PE is responsible for

$$s = \left\lceil \frac{|\mathcal{D}|}{n} \right\rceil$$

[1]At least in a relevance feedback environment this assumption is not quite correct, since the co-occurence of words in the documents of the database bears some information about the co-occurence of words in relevance-feedback queries.

words of the dictionary.

The current implementation computes a suitable value for n according to the following considerations:

1. n should be sufficiently large to enable the system to store each of the documents (even the longest ones) in one bucket,
2. the number of words s per dictionary partition should be less than 2^{16}
3. favour small values of n to bigger ones in order to reduce the communication overhead, caused by recombining the partial results of the distributed document parts.

The second point of the above list guanrantees, that each partition of the dictionary contains less than 2^{16} distinct terms (without loss of generality). Thus each PE has to handle terms with indices out of a index set of at most 2^{16} different terms. Consequently it is possible to store term indices in only 2 Bytes (one of those rare synergy-effects of parallel computing).

Typical values for n lie between $n = 2$ for the Virginia-collections (LISA, CISI, CRAN, ...) and $n = 8$ or $n = 16$ for the Tipster-database.

By now we only considered, how to balance the query-terms. Far more important is the distribution of document terms: The partitions of the database dictionary are assigned to same sized, disjunct sets of processing elements (cf. Figure 2.3). A necessary claim for achieving an evenly balanced load distribution is therefore, that the total number of references (terms of the document collection) is roughly the same for each partition of the dictionary.

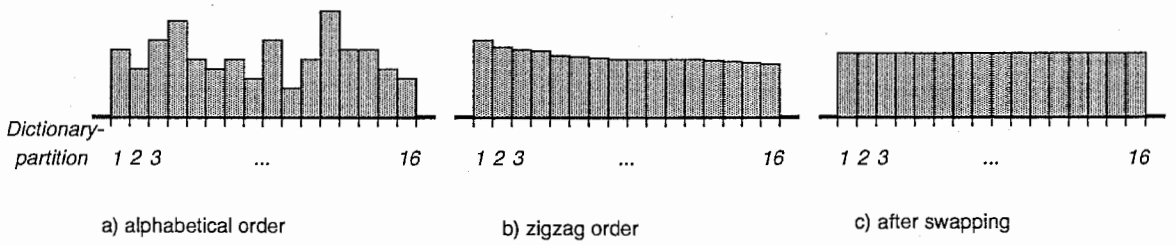

Figure 2.5: Number of terms per dictionary partition using different load balancing methods (LISA-collection, 16 partitions).

As figure 2.5.a) suggests, it is not a very good move to simply slice the alphabetically ordered dictionary in same sized pieces (pieces with the same number of words). Much better results are achieved by ordering the words according to their reference frequency and assigning them in a zigzag-manner to the dictionary partitions (fig. 2.5.b). The distribution can further be improved by swapping words with different frequencies between the partitions (fig. 2.5.c)[2].

Document Placement

The previous chapter described how the terms of documents and queries are distributed on the processing elements of a bucket. Since each database section is typically stored on

[2]An even better method would be to analyze the co-occurence of words in the document collection and assign words that occur oftenly together to *different* dictionary partitions. The performance gains are however unlikely to be significant.

more than one bucket of the PE-array, it remains to be addressed, where (i.e. on which bucket) to place documents.

Again the documents have to be allocated in such a way, that

1. the number of documents per bucket is roughly the same for each bucket,

2. the maximum number of terms per PE is as small as possible.

The number of buckets per section of the database depends on whether the section in respect is one part of a unclustered database, in which case all buckets of the PE-array are used for storing a section (in case of the TIPSTER-collection: 1024 buckets), or a cluster of the clustered one. In the latter case the number of buckets per cluster is typically significantly smaller (Tipster: 8 or 16 buckets per cluster).

It suggests itself to use the same kind of zigzag-allocation technique to distribute the documents on the buckets, as the one used for partitioning the database dictionary. However, while showing good results in partitioning the dictionary, it can only be used to generate a fast initial solution in document placement (cf. 2.1.1).

This initial distribution is refined by swapping documents between different buckets with the goal to reduce the maximum number of terms per PE. This is not as easy as in the case mentioned above (s. 2.1.1), where we just swapped words between different dictionary-sections. Now each swap not only affects two partitions, but all PEs of two participating buckets. If a swap reduces the number of terms on one PE of a bucket it may well increase the number of terms on the others. Figure 2.6 shows an example of two buckets ($n = 4$ PEs) filled with 3 documents each.

In the original distribution the processors 0 and 3 contain the same maximal number of document terms. Swap a) reduces the number of terms in these two, but increases the number in PE 7 to a new, higher maximum load. Swap b) moves the maximum load to PE 7, swap c) reduces it.

In the CRISP the following two methods to find a sequence of 'improving' document swaps are implemented:

1. greedy:
   as long as there are 'improving' swaps, one bucket is selected, that contains a processing element whose load is maximal in the set of all PEs (in the example: bucket 1) This bucket offers its documents one by one to all other buckets. Those buckets test, whether they have a document to give in exchange by which either the maximum number of terms over both buckets, or at least the number of processing elements containing the maximum number of terms is reduced (in the example swap b) and c) would be considered to be improving steps). The best swap is selected and performed.

   The algorithm terminates since the maximum number of terms per PE is monotonically decreasing, or while not changing, the number of maximum PEs decreases strictly.

2. probabilistic:
   swap documents randomly between buckets. Accept those swaps, that improve the load balancing of the two affected buckets.

   Terminate if no improving swaps occur within a sequence of randomly chosen swaps of pre-defined length.

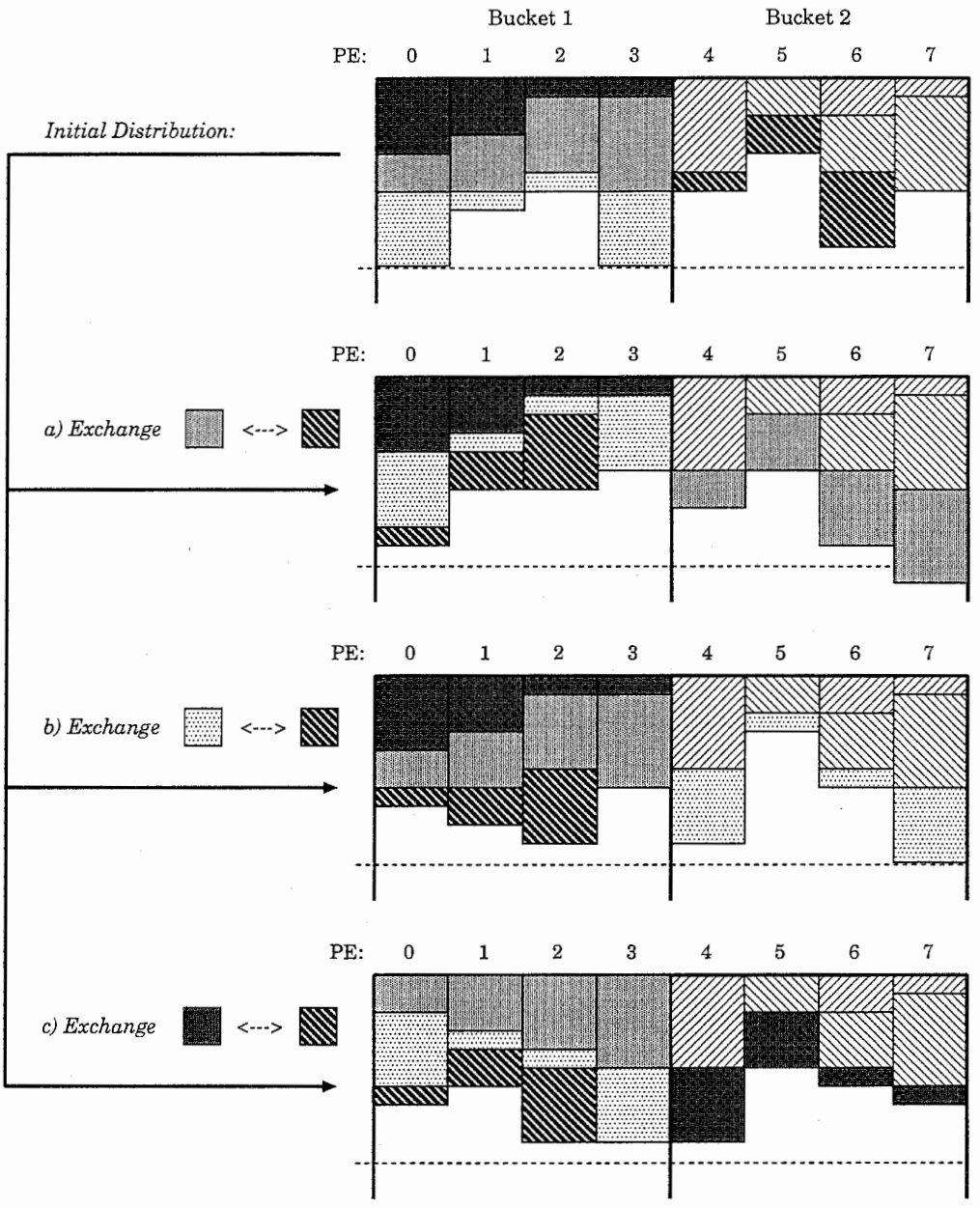

Figure 2.6: Swapping documents between two buckets.

Neither of the two algorithms can guarantee to find an optimal solution, but both perform well on the examined databases (cf. 2.1.1).

The running time of the greedy-algorithm is however for each swap quadratic in the number of documents per bucket and it may take some time to compute the document allocation for big document collections (even so it is implemented on the MP-2).

Therefore the CRISP-system uses by default the probabilistic allocation-algorithm.

Load Balancing Results

The following tables summarize the load balancing results for the LISA-database, the unclustered Tipster-database ('ucl') and the clustered Tipster-database ('cl'). The latter two differ in the size of the database-sections (ucl: full database, cl: average over 2000 clusters) and the number of buckets per section (ucl: 1024, cl: 8).

Results are given a) using a alphabetic dictionary partitioning b) the balanced dictionary partitioning of 2.1.1. The term-average gives a bound for the best possible, i.e. even distribution of terms on processing elements. The 'dumb'-distribution results from assigning document i of the database to bucket $i \bmod b$, where b denotes the number of buckets per section.

	term-av	dumb	greedy	prob.
LISA	27	49	31	32
TIPSTER (ucl)	6107	7956	—	6887
TIPSTER (cl)	391		—	

Table 2.1: Load balancing results with alphabetic dictionary partitioning

	term-av	dumb	greedy	prob.
LISA	27	46	28	30
TIPSTER (ucl)	6107	7278	—	6147
TIPSTER (cl)	391	480	—	427

Table 2.2: Load balancing results with balanced dictionary partitioning

The maximum load per PE of the probabilistic allocation algorithm lies for all observed databases maximal 10% over the one of the best possible distribution (evenly distributed document terms).

2.1.2 Query Computation

The data-structures described above are awkward, compared with the ones required for a sequential or the KSR-implementation of the vector space model. But this is not due to the SIMD-execution model of the MP-2, but they grew compicated because of the huge number of processing elements and their relatively small local memory capacity. The following section will concentrate on the peculiarities of the scoring routines caused by the SIMD-execution model of the MP-2.

Using the data-structures described above, the ranking of a already loaded part of the database can be computed as follows:

1. distribute the query according to 2.1.1 on the PEs of a bucket
2. broadcast distributed query to the PE-array
3. compute the partial document scores on all PEs
4. sum up the partial results for all documents
5. rank results

The distribution of query terms on the PEs of a bucket has been described in 2.1.1. The communication overhead of the interprocessor communication steps 2 and 4 is negligible (the data structures have been designed so as to allow them to be carried out by using the extremly fast local communication structure of the Maspar).

Point 3 will be discussed in detail below, because these routines form the core and most time consuming part of the retrieval system and reflect the peculiarities of SIMD-programming.

The ranking routines (5) are only mentioned in so far, as they restrict the computational scope of the retrieval system in respect to relevance feedback methods.

Computation of Document Scores

Sparse vectors are usually stored as a list of index/value-pairs containing all non-zero terms of the vectors. The inner product between two sparsely filled vectors $\vec{q}$ and $\vec{d}$ in this representation is on sequential and MIMD-machines usually computed by merging the sorted lists of indices:
I:

Notation:

$\|\vec{d}\|$	number of non-zero terms in $\vec{d}$
$\mathrm{I}(\vec{d}, n)$	index of n'th term in $\vec{d}$
$\mathrm{V}(\vec{d}, n)$	value of n'th term in $\vec{d}$

Input: sorted index-representation of vectors $\vec{d}$ and $\vec{q}$

```
i_d = 0
i_q = 0
result = 0
while ( i_d < |d| and i_q < |q| )
    {
        if ( I(d, i_d) < I(q, i_q) ) i_d++
        if ( I(d, i_d) > I(q, i_q) ) i_q++
        if ( I(d, i_d) = I(q, i_q) )
            {
                result = result + V(d, i_d) * V(q, i_q)
                i_d++
                i_q++
```

```
        }
    }
Output: result = d⃗ × q⃗
```

Complexity of the algorithm: The number of multiplications ('hits') depends on the proportion of matching terms in the two vectors. It is bounded by $\min(|\vec{d}|, |\vec{q}|)$. One might expect it to depend in a quadratic way on how sparsely filled the vectors are (e.g. if both vectors contain evenly distributed 1 % of all possible terms with non-zero weight, only one of 10000 possible 'hits' would occur statistically). This is however not true for relevance feedback: even so both query an document vectors contain a very small percentage of terms with non-zero weight (e.g. LISA-collection: documents 0.3%, queries typically between 1.5% and 3%), Table 1 (chapter 1.1) states, that the number of multiplications can be assumed in a relevance-feedback environment to lie between $0.3|\vec{d}|$ and $0.5|\vec{d}|$.

Thus using this algorithm, on MIMD and sequential machines the computation of the inner product between two vectors takes $|\vec{d}| + |\vec{q}|$ index operations and $0.5|\vec{d}|$ multiplications.

On SIMD-Machines however some thousand processing elements execute the above code simultaneously with different data. Chances are, that at each step some PEs want to proceed in one, some in the other vector and others have encountered a 'hit'. The code complexity is still $O(|\vec{d}| + |\vec{q}|)$, but in each step a SIMD-machine executes all three if-cases including the case of a 'hit', thus leading to $4(|\vec{d}| + |\vec{q}|)$ index operations and $|\vec{d}| + |\vec{q}|$ multiplications.

Moreover in computing the product between one vector $\vec{q}$ and a set of vectors $\{\vec{d}_i\}_i$, the (in the case of document retrieval usually relatively long query-) vector $\vec{q}$ has to be traversed once for each (relatively short) $\vec{d}_i$ (if one wants to avoid intermingling terms of different documents, which would lead to some memory overhead).

A better way of vector multiplication using a SIMD-architecture is to extend one of the sparse vectors ($\vec{q}$) to its full vector representation, i.e. to an array containing also the zero-terms of the vector. The indices of the non-zero terms of the second vector $\vec{d}$ can then be used to address the corresponding terms in the full vector $\vec{q}$:

II:

Notation:
q[i] term i of extended vector $\vec{q}$

```
Input: (un-sorted) index-representation of d⃗
       full-vector representation of q⃗
result = 0
for ( i_d = 0 ; i_d < |d⃗| ; i_d ++ )
    {
        result = result + V(d⃗, i_d) * q[I(d⃗, i_d)]
    }
Output: result = d⃗ × q⃗
```

This algorithm requires $|\vec{d}|$ multiplications for computing the inner product between two sparse vectors, independent of the number of non-zero terms in $\vec{q}$. The conversion of $\vec{q}$

from index-list to full-vector representation requires $O(|\vec{q}|)$ operations, but has to be done only once in computing the inner product between one query vector and a set of document vectors.

Algorithm II is suitable for SIMD-architectures since all processing elements execute exactly the same instructions at a time, independent of the structure of the sparse vectors.

If each processing element contains exactly one document (respectively the part of one document), the running time of algorithm II on a SIMD-machine will be

$$T_1 = O(|\vec{q}|) + t_m * \max_{pe}(|\vec{d}_{pe}|)$$

Where t_m denotes the execution time of the loop-body of algorithm II.

If parts of more than one documents are computed by each processing element, in each loop-iteration some processing elements might change from one document to another. Let t_c denote the time for transition from one document to the next (i.e. switching the variable used for accumulating the result), then the running time of the above algorithm with p documents per PE is

$$T_p = O(|\vec{q}|) + (t_m + t_c) * \max_{pe}(\sum_{j=1}^{p} |\vec{d}_{pe,j}|)$$

Algorithm II has however one major drawback: the dictionary of distinct words of a real-world database might well contain some hundred thousand words, making it infeasible to store the full-vector representation of $\vec{q}$ on the PE-array.

So the full vector representation is replaced in the CRISP-system by a hash-table for the non-zero terms of the query, leading to algorithm III:

III:

Input: (un-sorted) index-representation of $\vec{d}$
hash-table representation of $\vec{q}$: h_q
result = 0
for ($i_d = 0$; $i_d < |\vec{d}|$; i_d++)
{
$result = result + V(\vec{d}, i_d) * h_q[\,\text{hash}(I(\vec{d}, i_d))]$
}
Output: result = $\vec{d} \times \vec{q}$

The formulas given for the execution time of algorithm II hold unchanged for algorithm III, with a slightly bigger time constant t_m. The current implementation uses a modulo-hashing function with a power of 2 as hash table size (i.e. the hashing function is a projection on the lower bits of query-term indices). Collisions are resolved by linear progression in the hash-table.

Document Ranking

The output of the CRISP-retrieval kernel for a given query consists of a list of the highest scored documents, the length of which depend on the users need of information. The CRISP-system does usually not return a list of the scores of all documents to the calling instance.

One raw ranking of the document scores is however not enough to meet the requirements of a relevance feedback system. The IDE-DEC-HI formula requires

- the ranking of the top scored not yet retrieved documents
- the top ranked non-relevant document that hasn't been used by now for reformulating a query. independent of whether it has been retrieved in this or in prior relevance feedback iterations.

In order to meet the needs of as wide a range of relevance feedback methods as possible, the CRISP ranking routine returns two ranking lists and discerns three types of documents:

1. documents that are ranked in list 1
2. documents that are ranked in list 2
3. documents that are not included in either list

For the IDE-DEC-HI-formula, list 1 consists of the ranking of the d highest scored, not yet retrieved documents. List 2 of the one highest ranked non-relevant document of prior iterations, that hasn't been used for query-reformulation. All documents that have been used for query-reformulation are not ranked in further iterations.

The retrieval kernel should thus be applicable for most relevance feedback methods (e.g. the Rocchio-formulas, Ide-regular, Common-term-system).

2.1.3 Summary

This chapter described

a) the data structures and load balancing routines of the parallel retrieval kernel of the CRISP-system

b) a SIMD-algorithm for scoring the documents

The load balancing results lay for the Tipster-database for all observed distributions within 10% from the optimal solution. The communication overhead, caused by interprocessor communication during query-processing is negligible.

The data-structures have been chosen in a way, that makes it possible to formulate the multiplication of sparsely filled vectors in a pure SIMD-manner. Due to the typically high proportion of matching terms in query and document vectors, the program complexity of this algorithm is comparable to the one of the List-Merging algorithm used on sequential or MIMD-parallel machines.

Thus it has been possible to formulate the retrieval phase of the VSM-model in a pure SIMD-fashion. However it has to be noted, that the complexity of the required code and data-structures lies far beyond the one for comparable MIMD or sequential implementations. This reduces the changeability and therewith the flexibility of SIMD-systems considerably and is a major drawback of the presented approach.

2.2 Cluster-Subsystem

The task of the cluster-subsystem of CRISP is to restrict the search scope of the full-database search, without disturbing the search process, i.e. loosing the retrieval effectiveness of the VSM. In order to do so it has to

1 cluster document collections, assigning documents that are to be relevant to the same queries in the same clusters (of course without knowledge of the actual query),

2 identify those clusters that are most likely to contain relevant documents for a given query.

The relevance of documents to queries, used to formulate the two goals above, is actually a very vague concept, especially as long as the queries are unknown. So for clustering purposes we accept the relevance estimations of the VSM-retrieval system:

1' cluster document collections, assigning documents with the same response behaviour under the VSM to the same clusters

2' identify those clusters, that are most likely to contain the documents, that would occur in the top-ranked documents of a corresponding full-database search

Section 2.2.1 is concerned with the clustering method chosen in the CRISP-system. One important property of this clustering method will be, that the resulting system can use the same parallel retrieval kernel for clustering purposes (2.2.2), for selecting the most promising clusters of the database (2.2.3) and for ranking the documents within these clusters.

2.2.1 Clustering method

The clustering algorithm has to meet the following requirements:

1. it has to be precise enough to identify for a given query a (small) subset of clusters that contain a 'sufficient' proportion of relevant documents.

2. its time and space complexity has to be small. Methods with space-complexity over O(N), where N denotes the number of documents, are clearly not feasible for databases with probably millions of documents.

3. The resulting clusters should be of the same size, in order to be computed efficiently by a SIMD-machine.

   In order to keep the storage requirements for the cluster centroids reasonably small the cluster structure will have to be coarse.

4. The algorithm should be stable under small changes in the database (e.g. adding more documents to the database).

Unfortunately, the clustering mechanisms that showed best results in terms of retrieval effectiveness (1.) and whose cluster hierarchy tends to be balanced (3.) (complete link, HBC), are prohibitively expensive to be used for big document databases.

Moreover, even if one would be able to compute the cluster hierarchy for big databases, a SIMD-system as the one described in this paper could not profit much from this hierarchy: SIMD-machines favor broad, extensive searches (and offer the required computing power to do so), while their performance on tree-searches is comparatively poor.

On the other hand, non-hierarchical methods, which meet requirement 3. by definition, are relatively inexpensive to compute (2.) and reasonably stable under changes in the database (4.). Therefore we chose a non-hierarchical re-allocation method for the clustering subsystem of CRISP. Experiments will have to clarify its suitability in the current setting with respect to point 1. of our above requirements.

The clustering algorithm of CRISP

The clustering subsystem of CRISP uses the following re-allocation method to compute a non-hierarchical cluster structure of document collections:

```
1       # given the number of documents per cluster: n
2       # (and thereby the number of clusters)
3       initialize: distribute documents randomly on clusters,
4                   compute cluster centroids
5       WHILE not stable
6                   {
7                   randomize order of clusters
8                   FOR each cluster
9                       {
10                      look for the 2n documents with highest cluster-centroid-similarity
11                      take the n best documents into the cluster, that are
12                          -either not yet assigned to any
                             cluster in this iteration
13                          - or have a higher similarity to this centroid
14                           than to the centroid of its current cluster
15                      }
16                  force unclustered documents in incompletely filled clusters
17                  compute cluster centroids
18                  }
19      output:     cluster assignment
```

The choice of the cluster representatives (line *4* and *17*) turned out to be crucial in order to avoid premature convergence of the clustering process: in using the normalized sum of the document vectors, a cluster centroid will contain, among others, all the unique terms of its documents. Those terms add only to the score of the documents that are already in the cluster, without carrying any information about the interrelation between them, making the clustering process very conservative. So we only use those terms in the clustering process, that occur in at least two documents of the cluster.

Furthermore it proved worthwhile to truncate the centroid vectors in early iterations to the words with highest weight in order to overcome the bad initial distribution fast. The number of words in the cluster centroid is then gradually increased over the iterations.

2.2.2 Cluster generation

The above method is actually nothing else than a variant of the *k-means-algorithm*. It differs in the following points:

- all clusters contain the same number of documents
- an already assigned document can be re-assigned to a better fitting cluster in line 10. The cluster from which the document is taken cannot select a new document in the same iteration. Thus the distribution of documents on clusters in lines 8-14 is incomplete.

We already mentioned the reason for the first modification of the k-means-algorithm (requirement 3.). Its quite unnatural formulation and the incomplete assignment of documents on clusters follows from how the cluster subsystem is embedded into CRISP and will be explained below.

The usual way to describe the k-means-algorithm is something like this: "assign each document to the cluster with highest document-centroid similarity, iterate this process". Thus the algorithm computes for one document its similarities to all clusters-centroids, assigns the document to the best fitting cluster and continues with the next document.

In contrast the algorithm as formulated above takes one cluster, computes the similarity of one centroid vector to all documents and assigns the best-fitting n documents to this cluster.

This is an important difference because the parallel retrieval kernel is a highly efficient tool for computing exactly this (using the centroid vector to query the database).

At this stage of the database generation there usually exists only the unstructured collection of document vectors as they are returned by the vector-generator. In order to be used by the parallel retrieval kernel during the clustering process, the original collection has to be translated into an intermediate, distributed representation (Figure 2.7).

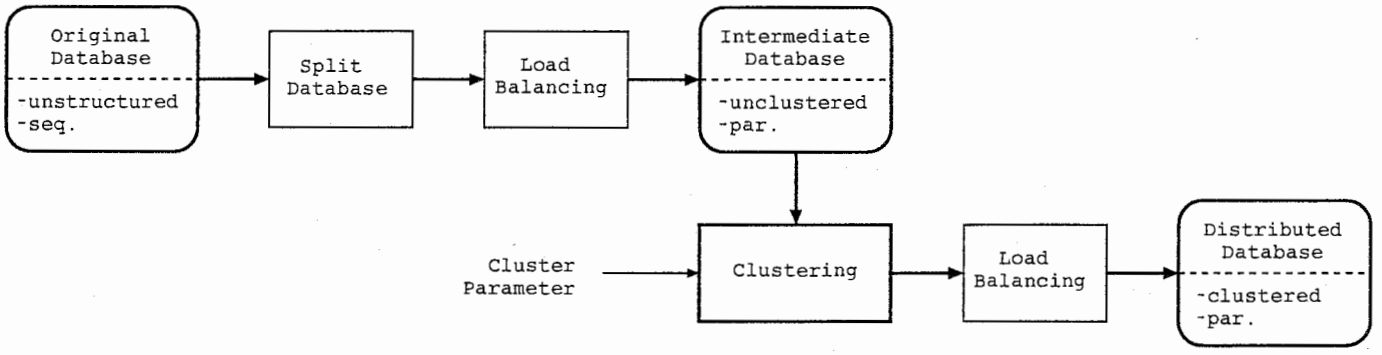

Figure 2.7: Generating the clustered, distributed database.

The database is divided into several sections, each of which is small enough to fit completely in the local memory of the PE-array. The parallel data-structure for this sectionized database is computed using the same load balancing routines, that are applied afterwards to generate the final, clustered database. Of course it would not make sense to use a cluster search on the intermediate database, but it can be used for the full-database searches performed during the clustering process.

The clustering routines operates on this intermediate database. The discussion of the clustering process left the choice of clustering parameters, e.g. the number of clusters and the number of terms per cluster centroid, open. Suitable values have to be determined experimentally (A.1).

The IO-overhead caused by transferring very big databases in multiple parts to the memory of the PE-array is no serious problem in the full-database searches required for clustering a database, since several hundred or thousand queries (centroid vectors) have to be processed at the same time. After loading one part of the database, the partial results for all queries can be computed for this part and be merged with the results on prior parts before the next part of the collection has to be loaded.

2.2.3 Cluster search

The search process in the clustered database consists of the following two parts

1. identify the most promising $r\%$ clusters,

2. load clusters,

3. perform a regular VSM-search in those clusters.

The CRISP-system uses the same cosine-measure for computing the similarity of the query to the cluster centroids as is used for ranking the document collection according to the query. So both stages turn out to require the same computations, namely the ranking of a vector collection according to their similarities to a given query-vector. It does not matter, whether the collection consists of document vectors or of cluster centroids. Since the centroid collection has to be queried once for each user-query, it is stored permanently in the PE-array.

Thus we can use the same retrieval kernel for clustering databases, the identification of the most promising clusters and for retrieval purposes within the selected part of the database.

Loading the top-ranked clusters

The percentage of the database r that is inspected during query processing is a user defined parameter. The best $r\%$ clusters are selected as described above.

The loading of these clusters poses however yet another load balancing problem: By now we used the term 'same sized' for clusters with the same number of documents. Now the system has to load a set of selected clusters and place them in a manner on the PE-array that balances the load, i.e. the number of terms, not the number of documents.

Documents from different sources, treating different subjects tend to differ considerably in their length. The clustering process is considered to collect documents related to the same topic in the same clusters. Thus one can expect the size of clusters containing the same number of documents to differ considerably if measured in the number of terms per cluster. In the example of the Tipster-collection: after stripping all clusters with exeptionally few document terms, (they occur so infrequently that they can be ignored for load balancing reasons), the number of terms per cluster differs by a factor of approximately 10.

Moreover it is advantegeous to store – if possible – all clusters of the current search scope at once on the PE-array (see 3.2). This means that for small r, the documents of each cluster will have to be stored on a relatively big number of processing elements, while the number of buckets is small for big r.

The cluster-search routine has to solve the following load balancing problem at run-time: find an evenly balanced allocation for a given set of clusters on the PE-array, where

- the number of cluster depend on the run-time defined parameter r
- and the cluster size is inhomogeneous.

Load between two consecutive queries (e.g. two relevance feedback iterations) only the part of the database, that is not already present on the PE-array.

The load balancing routines on document level described in 2.1.1 are too time consuming for online-use. The current implementation of the CRISP distributes the documents of each cluster during the generation of the clustered database on a pre-defined number p of pieces using the load-balancing routines of 2.1.1. p has to be selected to be big enough to balance the load betwenn different sized clusters (in the Tipster collection a 1:10 ratio, i.e. $p \geq 10$) and to allow the exhaustive use of the full PE-array even for minimal r. For

example: the Tipster-collection is distributed on 2000 clusters, the PE-array subdivides in 1024 buckets. The minimal sensible load is asumed to be $r = 5\%$ or 100 clusters. In order to fill all buckets, each cluster has to be divided on at least $p = 10$ pieces.

The database used for retrieval purposes is now structured as follows:

- The document collection divides into *clusters*.
- Each cluster subdivides into p *pieces*,
- containing roughly the same number of *documents* and document terms.
- The *terms* of each document are distributed on the PEs of a bucket.

The cluster-loading algorithm first removes all clusters from the PE-array, which are not in the set of required clusters $\mathcal{R}$ and all clusters that are already present on the PE-array from the set of required clusters. The load-balancing goal is to distribute the terms of the reqired clusters evenly on the unoccupied buckets. The average number of terms per bucket g is computed depending on the number of unused buckets of the PE-array and the number of terms in the set of still required clusters. Iteratively one of the demanded clusters is selected and its pieces are distributed on so many buckets, that the load per bucket slightly exceeds the load-balancing goal. The respective cluster is removed from $\mathcal{R}$, g is recomputed and the next cluster is selected.

This allocation algorithm is far from being optimal, but produces reasonable results in O(#clusters) time.

Chapter 3

Results

The clustering-subsystem of the CRISP modifies the search-behaviour of the regular VSM by restricting its search scope to a small portion of the database. Consequently the chosen approach has to be justified examining both relative retrieval effectiveness and efficiency compared to the exhaustive search of the regular VSM.

3.1 Effectiveness

3.1.1 Measurement method

Since the vector space model with relevance feedback is a well-known retrieval method we concentrate in the following investigations on the relative effectiveness of full-database and cluster-search. Our current parallelization does not affect the behavior of the standard-VSM, and thus we can take the recall/precision of this system on the full database (i.e. without clustering routines) as a reference system.

The effectiveness of a retrieval system is usually measured by considering a recall-precision-graph, showing the interdependance of recall and precision. Those graphs are obtained by evaluating the retrieval precision and recall at different levels of retrieved documents. Instead we consider the recall or the precision at only one document level, i.e. after a given number of documents have been retrieved and examine how these values depend on the percentage of the database used as search scope for the cluster-search (there is no need to consider both values, because they depend proportionally on each other for given document level and test set [Voor86]).

The change in recall/precision due to the search scope restriction measures the effect of the cluster search on the regular search process. This measure might however be too abstract to gain much insight in the clustering process, because it gives only one combined evaluation for the regular search process, the feedback operations and the cluster quality.

Figure 3.1 shall clarify the differences in the outcome of a full-database search and a corresponding cluster-search: the relative order of the documents in the ranking list of the cluster-search is the same as in the one of full-database search. Differences in the results of both methods occur only, if one or more of the top-rankend documents of the full-database search do not occur in the top-ranked clusters of the cluster search (in the example documents no 7 and 8).

The disturbance of the full-database search process due to the restriction of the search scope can thus be measured more directly by observing the percentage of documents, that are retrieved by the reference system, but that do not occur in the top ranked clusters of the restricted search for the same query. We call this percentage the '*agreement*' of the

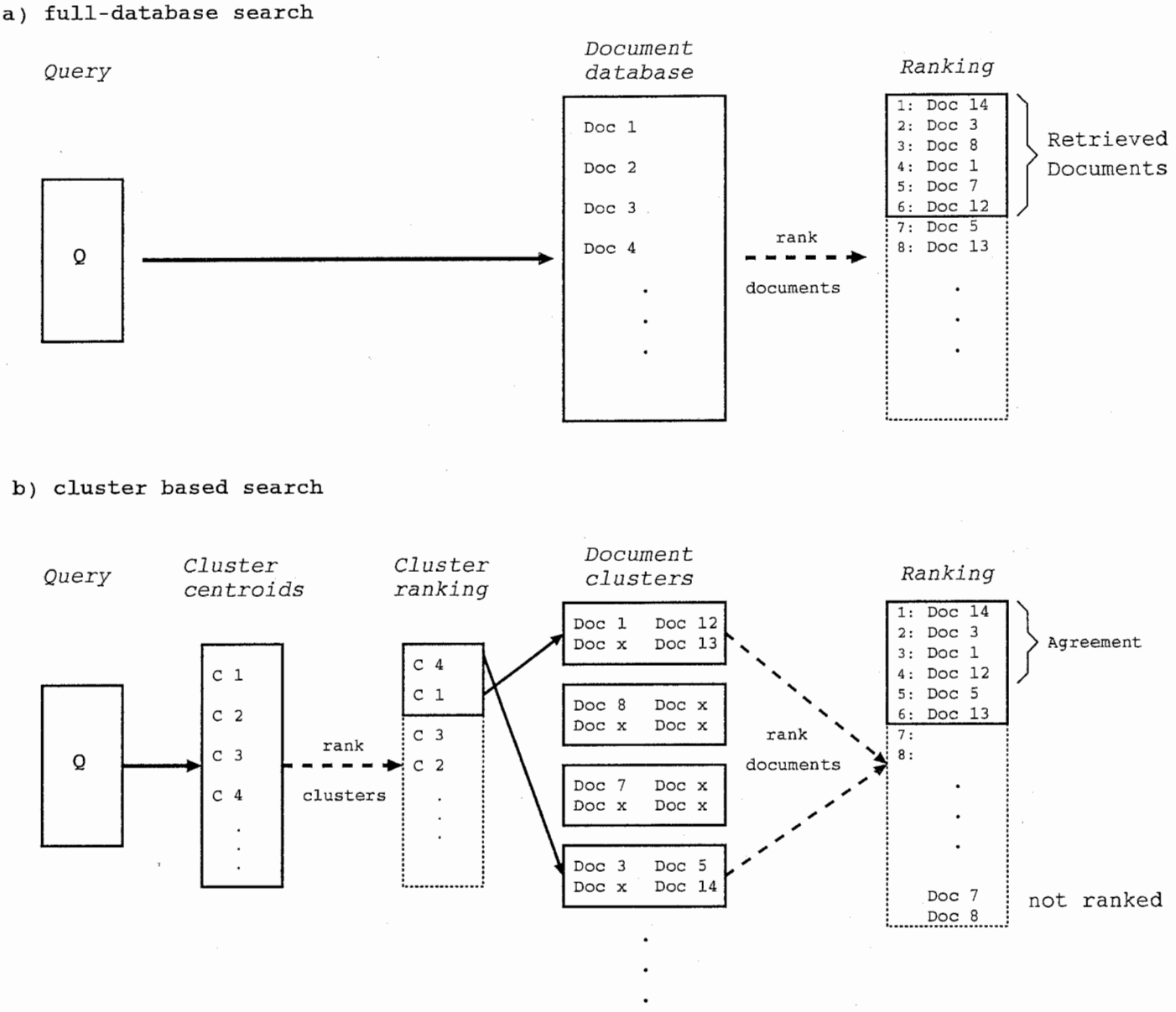

Figure 3.1: scheme of a) full-database search b) cluster-search.

two search methods[1].

The agreement does not translate directly in a proportional loss of recall/precision, since the place of missing top-ranked documents is occupied in the cluster-search by other highly ranked documents (in the example documents no 5 and 13), that might be relevant to the given query too.

All measurements have been made performing 8 relevance feedback iterations for each query of the standard test set, independent of whether preceding relevance feedback iterations found relevant documents or not. In each iteration 20 documents were retrieved.

[1] An alternative measure for the cluster quality can be derived by observing for all clusters the average of the full-database search-ranks of all cluster members. The average rank of documents in top-ranked clusters (now in cluster-search mode) should be significantly lower than in lower ranked clusters.
The results of this measure were in all observed cases consistent with the agreement measure.

3.1.2 The Effectiveness of the CRISP-system

Different databases have been shown to respond very differently to clustering approaches. Thus the performance of the CRISP-system is evaluated on three different standard test sets of the Virginia-collections (Cranfield, CISI, LISA) and disk 1 of the Tipster-database. Of those the Cranfield collection has been noted in the literature [Will88] to be quite well-disposed in respect to document clustering, while the contrary holds for the CISI-collection [Voor85].

Figures 3.2 and 3.3 show the influence of the search scope restriction on the retrieval effectiveness of the CRISP-system, in dependency of the percentage r of the database to which the search scope has been narrowed down. For LISA, CISI and Cranfield the number of terms in the centroid collection amounts to approximately 5% of the respective database-size, for Tipster approximately 2%[2].

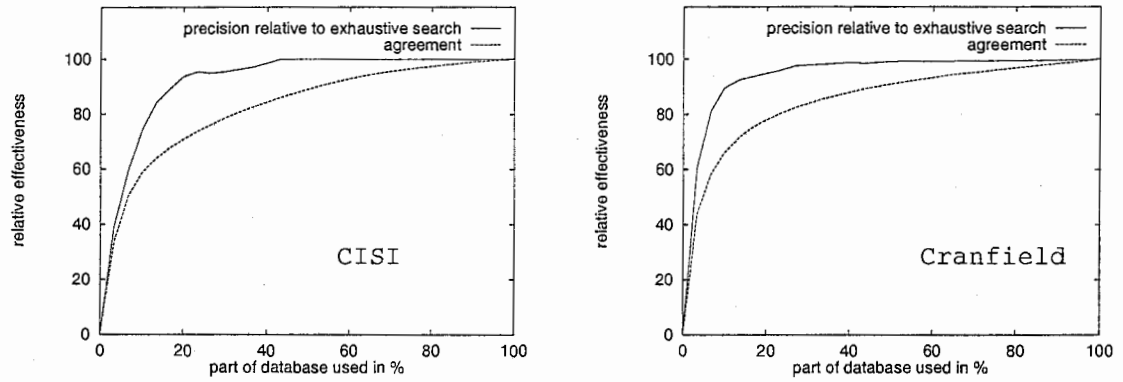

Figure 3.2: Average relative effectiveness and agreement with full database search of cluster search for (a) CISI and (b) CRAN

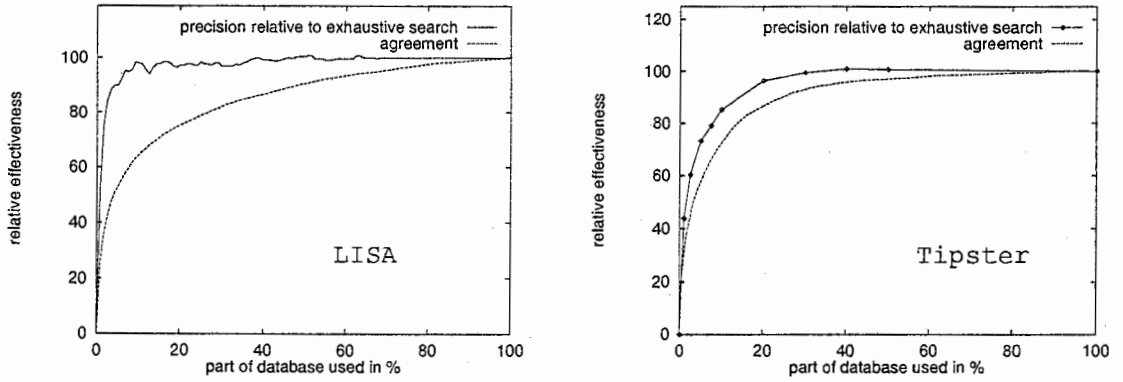

Figure 3.3: Average relative effectiveness and agreement with full database search of cluster search for (a) LISA and (b) Tipster

The relatively high agreement between the two search processes, i.e. the percentage of documents that are retrieved for the same query by both the cluster search and the unrestricted search for small r (e.g. at $r = 10\%$ the agreement is approximately 70%) indicates, that

[2]LISA, Cranfield, CISI: 50 documents per cluster, centroids truncated to highest weigthed 100 terms
Tipster: 250 documents per cluster, centroids truncated to highest weigthed 500 terms

- the adopted clustering method has been able to assign documents, that are retrieved by a unrestricted search process (in average over all queries of the test-set) to the same clusters.

  Moreover, the gap between relative effectiveness and agreement at low r shows, that the clustering process collected documents that tend to be *relevant* to the same queries in the same clusters.

- the cluster-search could detect those clusters in which they are concentrated.

In restricting the search space to 10 % of the database, the cluster search reached only 75% of the effectiveness of the exhaustive search on the CISI-collection, but 90% on Cranfield and 95% on the LISA-database. On the Tipster collection we measured a performance of 86%.

In order to get a relative effectiveness of over 95% we have to inspect 23 % of the CISI-database, 20 % of the Cranfield-database, 16 % of the Tipster- and 11% of the LISA-collection.

3.2 Efficiency

What are the benefits of the cluster-search for the retrieval efficiency of CRISP?

We have seen that it is possible to restrict the scope of the search process to between $r = 5\%$ and $r = 20\%$ of the database, depending on the precision-needs of the user and the database.

The selection of the concerning clusters is done by the same process as the search itself. Therefore the computation overhead of the cluster-search for ranking the cluster collection depends in the same way of the centroid collection size c (2% to 5%) as for a regular document collection.

The overall computational requirements of the search process is therefore reduced by the cluster search to $r + c$ percent of the cost of a full-database search (i.e. to between 7% and 25%).

Even more important is what we can gain in terms of reducing the amount of data that has to be transfered to and from a secondary storage (obviously it makes no sense to look at the small Virginia test-sets in this context).

Since a copy of the cluster centroids is stored permanently in the local memory of the PE-array, the IO-costs reduce to at least r % of the database or by a factor of 5 to 20 relative to a full-database search.

The above figure of r % is actually not quite correct, since a portion of the database will at each time be present in the local memory and thus does not have to be loaded for both search-methods. This changes, if the system has to compute a sequence of queries that were re-formulated by relevance feedback. Re-formulated queries tend to select the same clusters as the queries of prior feedback-iterations.

If it is possible to store the r% of the database that were used by prior iterations till the next re-formulated query arrives, the search process will mainly proceed in the already loaded part of the database.

Figure 3.4 shows the percentage of the database that has to be loaded for the user formulated query, respectively between two iterations of the relevance feedback process for $r = 10\%$.

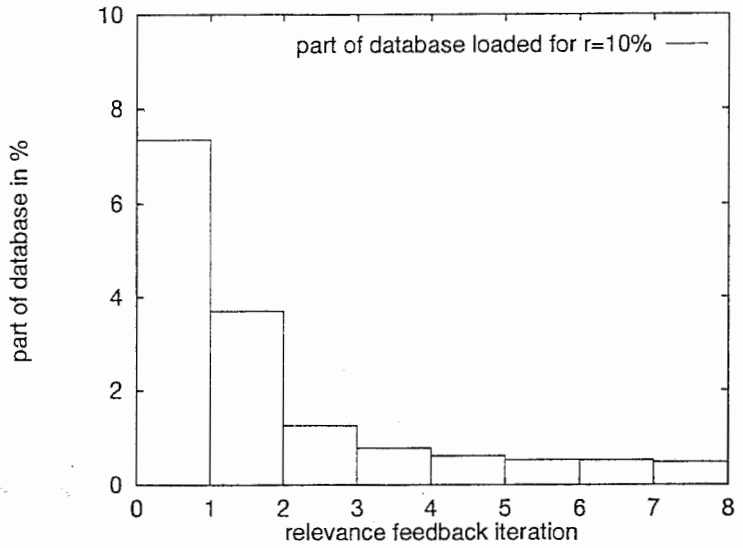

Figure 3.4: Tipster: percentage of database loaded between relevance feedback iterations ($r = 10\%$)

The percentage of the database that is searched by a re-formulated query, but not by its direct predecessor is on average for the Tipster collection only 0.8% ($r = 10\%$).

An example describing the performance of the CRISP on the Tipster-collection:
size of the centroid collection: $c = 4\%$
search scope (e.g.): $r = 15\%$
$\Longrightarrow$

relative retrieval effectiveness:	-7%
computational costs:	reduced to 17%
database loaded per query:	reduced to 2.8% of the full-database costs

Chapter 4

Conclusions and Future Work

The goal of our work has been to develop a likewise efficient and effective retrieval system for very large databases:

- We developed a parallel VSM-retrieval kernel on a SIMD-machine. The suitability of SIMD-architectures for this task has been validated by a) presenting effective load balancing routines b) formulating the retrieval process as a pure SIMD-task.
- Non-hierarchical clustering methods have the potential to cluster even very big databases with sufficient precision. With a simple variant of a k-means-algorithm it has been possible to reduce the search scope of an exhaustive VSM-search to between 11% and 23% of the database with a loss of only 5% in retrieval effectiveness. The size of the centroid structure used to obtain these results is with 2%-5% of the full database small.

The parallel retrieval kernel, clustering and cluster-searching routines based on this kernel form the CRISP. This system is efficient enough to handle even big databases like the Tipster-collection, while preserving most of the retrieval effectiveness of the original vector space model.

Future work has however to be aimed at the comparison of the chosen approach to other, more sophisticated clustering methods. How well would hierarchical methods behave in the same setting?

Appendix A

Implementation Details

A.1 Clustering behaviour

The standard clustering method of the CRISP is heuristic in nature. Its performance depends on the initial ordering of documents and several parameters that have to be determined empirically, most important: the number of clusters and the number of terms per cluster-centroid.

The following results describe the behaviour of the clustering subsystem on the Tipster and LISA-database. They are however far from being complete and for thorough understanding of the clustering behaviour further studies are certainly required.

Choice of Parameters

In discussing the clustering method of the CRISP the following points have not been addressed:

- the number of documents per cluster: n
- the length of the cluster representative (centroids): l
  (from centroid vectors containing more than l terms all but the heighest weighted l terms are removed)

One drawback of the heuristic clustering method of CRISP is, that suitable values will have to be determined experimentally.

For the Tipster collection, divided on 2000 clusters ($n = 250$ documents per cluster):

$n \setminus l$	50	100	200	500	1000	2000
250	69.7	69.7	72.8	72.8	73.1	74.3

Some findings:

- the agreement of full-database and cluster-search is higher using many small clusters, i.e. a fine grained cluster structure, than using a coarse cluster structure.
  (Experiments performed on Cranfield, CISI and LISA collection)
- the longer the cluster centroids the higher the effectivenes of the cluster search. The cluster-search shows however acceptable results even for very short cluster-centroids.

The collection of centroid vectors has to be searched once for each cluster-search query. This takes time linearly depending on the number of terms in the centroid collection. Thus

the size of the centroid collection measured in the number of non-zero terms has to be small compared to the size of the document database, leading to a trade-off between the two cluster parameters in determining suitable values for a given database.

An interesting and still open question is, whether suitable values for the number of documents per cluster are related to the size of the document collection. Experiments on the Cranfield, LISA and Tipster-collections indicate that suitable number of documents per cluster is not a constant for all databases, but is positively correlated with the number of documents in the database (i.e. that the number of clusters do not have to be increased linearly with growing document size in order to get the same level of retrieval effectiveness). The results are however not conclusive.

Convergence

It turned out to be worthwhile to use short centroids to represent the cluster content during the early iterations of the clustering process and to increase the length of the centroids slowly with the number of iterations. For clustering the Tipster collection, the centroid length l has been set in the i'th iteration to:

$$l = 30 + i * 5$$

Cluster centroids $\vec{t_c}$ are computed by truncating the mean of the cluster members $\vec{d_i}$ to the highest weighted l terms:

$$\vec{t_c} = \frac{\text{trunc}_l\left(\sum_{i \in C} \vec{d_i}\right)}{|C|}$$

In using this formula (it showed better results than the use of normalized centroids) it is hard to define a convergence-criterion for the clustering process, since the document scores (similarities to the cluster centroids) keep growing with increasing number of terms in the centroid vectors. Thus the cluster quality measured in the sum of the similarities of the cluster members to the cluster centroids keeps growing until the centroids are long enough to cover most of the terms of the cluster members (else-wise adding some more terms from cluster members to the centroid would result in an increase of the centroid-similarities of the respective documents).

By now this problem is overcome by checking periodically the agreement between the full-database search and a cluster search based on the intermediate clustering.

iteration	agreement
10	67.1%
20	70.0%
30	70.8%
40	70.4%
50	71.9%
60	71.2%
70	71.7%
80	72.8%

Table A.1: Agreement for $r = 10\%$ in dependance of the clustering iteration on the Tipster-database

Table A.1 shows the agreement between the full-database search and the cluster-search (restricted to $r = 10\%$ of the database) in dependance of the number of clustering-iterations performed on the Tipster-collection. The clustering process has been initialized by distributing the documents randomly on the database clusters.

Table A.1 indicates, that

(1) the agreement between both search processes reaches its final level in few iterations (for Tipster: approx. 20 iterations)

(2) but keeps increasing slightly for long

Cluster selection

The subset of clusters used as the search-scope of a cluster-search are selected according to the similarity between the cluster centroids and the query, measured in the same way as the similarities between documents and the query (usually by the cosine of the angle between each two vectors).

The suitability of this method can not be justified directly by observing the agreement-measure or the relative retirval precision, because both measures give only one combined evaluation for

a. the cluster quality,

b. the ability of the cluster search to identify the clusters that are most likely to contain relevant documents.

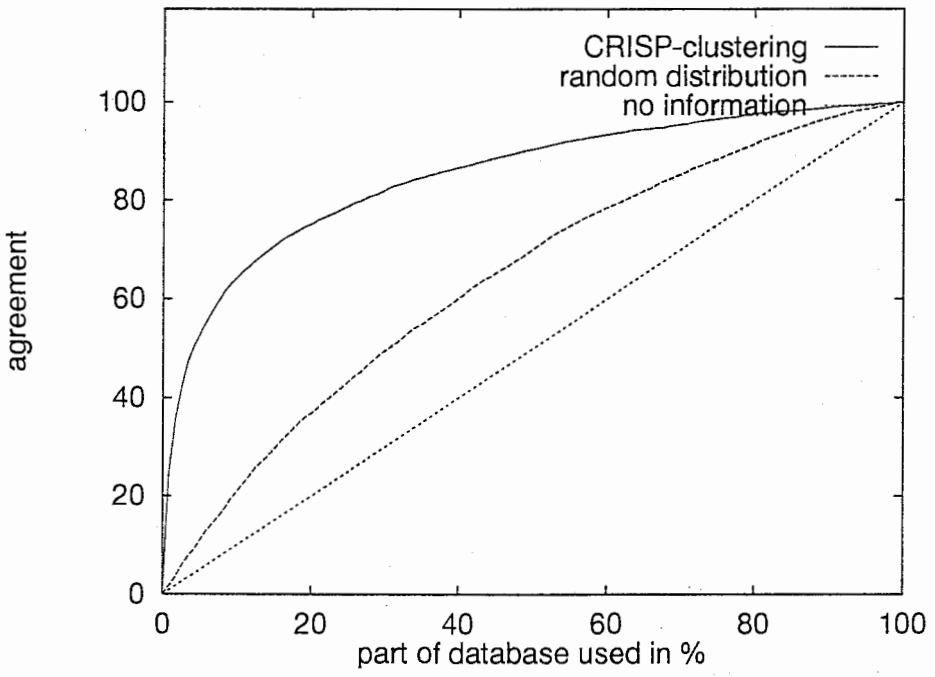

Figure A.1: Agreement of full-database and cluster-search on the LISA-collection.

Figure A.1 shows the agreement of a cluster-search on the LISA-collection depending on the percentage r of the database used as the search scope of the retrieval process. If documents are distributed randomly on the clusters of a database, one would expect the agreement to depend linearly on r (Figure A.1: 'no information'). But even for if one

distributes documentds randomly on the clusters one can expect the cluster-search to do better if the regular cluster-selection process is used, since the information stored in the cluster centroids will bear some diffuse information about the contents of the cluster. The 'random distribution'-curve of figure A.1 demonstrates the strength of this effect. The third curve shows the results of the clustering algorithm of the CRISP.

The first derivative of the CRISP-clustering curve (respectively its discrete equivalent) is strictly decreasing with growing r. Clusters, that contribute most to the agreement of cluster search and full-database search are highly ranked in the cluster-search. The cluster search process ranks the clusters on average over all queries in the same order, as would be obtained by ranking clusters according to the contained proportion of top-ranked documents.

We conclude, that the chosen cluster representation and cluster selection method enables the cluster-search process to identify on average over all queries the parts of a given cluster structure that are most likely to contain relevant documents.

A.2 Improving the clustering methods

The cluster quality could be increased significantly if it would be possible to identify those documents of the database during the clustering process that are not well represented by the current cluster centroids, i.e. that will not be found in a cluster search.

Let

t_i denote the number of queries, for which document i occurs in the list of top-ranked documents of the full-database search,

m_i denote the number of queries, for which document i occurs in the list of top-ranked documents of the full-database search, but not in the corresponding list of the cluster-search.

Then both the similarity of a document to the centroid of its current cluster (Figure A.2) and the fact whether the document is assigned to the most similar cluster or not bears some (weak ?) information about the proportion of failures $f_i := m_i/t_i$ in detecting the respective document.

A.3 Implemented Clustering Methods

The clustering algorithm described in 2.2.1 is the default method used by CRISP for being a) applicable for very big databases and b) effective. The following clustering methods are implemented in the CRISP:

- "Random"
  random distribution of documents on clusters
- "FeatureVector"
  Instead of using the document vectors for clustering, use a *feature vector of lower dimensionality* that describes the retrieval-behaviour of the document. Assign documents with the same retrieval-behaviour (i.e. that receive high respectively low scores for the same queries) to the same clusters.

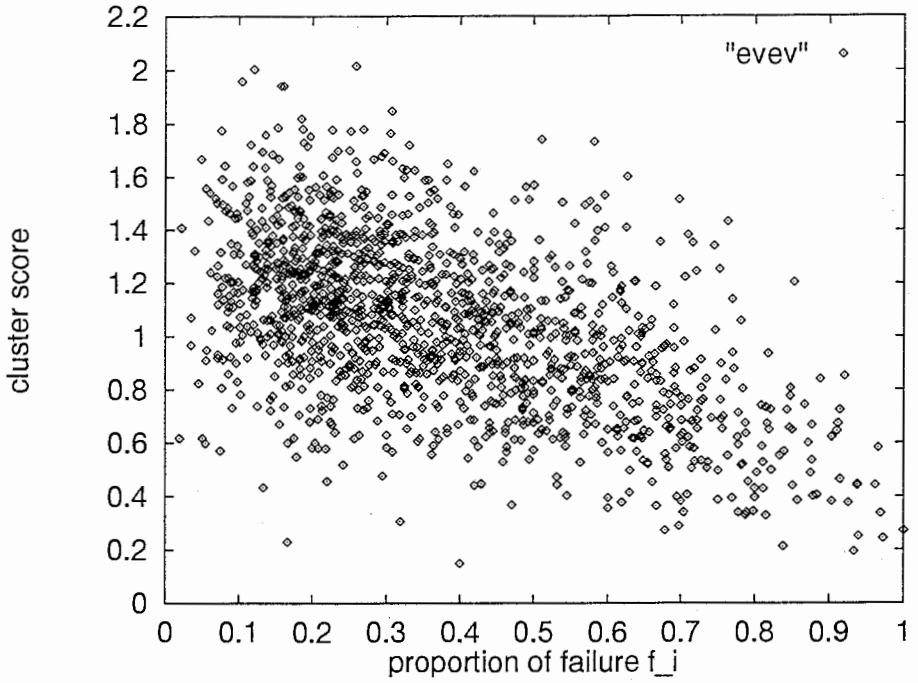

Figure A.2: Dependance of failure proportion on the centroid/document similarity.

The current implementation uses the scores of documents for f sample queries as feature vector of dimensionality f. One estimator for the structure of a relevance feedback query are the document vectors of the collections (a relevance feedback query is a linear combination of the original query and zero, one or more document vectors). Therefore f randomly chosen documents are used as sample queries to generate the feature vectors for the document collections.

The resulting vectors are clustered with a k-means algorithm.

This approach showed good results in terms of retrieval effectiveness only for manually selected sample documents or for long feature vectors.

- "Variant_I"
  "Variant_II"
  Pose randomly chosen documents as queries to the document collection. Assign the top-ranked documents to the same cluster.

  Iterate with the sum of the highest ranked n documents as new query ("Variant_II").

  The main advantage of this method is, that it does not require the computation of the cluster centroids in each clustering iteration.

  The effectiveness of this approach depends heavily on the selection of seed-documents and the order in which they are used for querying. It is suitable for generating fast initial clusterings.

- "Variant_II+"
  Clustering method as described in 2.2.1.

- "MPstandard"
  Variant of the k-means algorithm, that avoids the incomplete assignement of documents to clusters of the default clustering method (2.2.1). Most effective method

but its memory requirements (approx. 100 byte per document) are too high to be used to cluster the Tipster-collection.

Appendix B

Statistics on Document Collections

B.1 Statistics

Statistics of the document collections considered in this work, namely

- the Virginia collections CISI, CRAN and LISA
- Disk 1 of the Tipster-collection, consisting of 5 subcollections:
  - ap:
    stories from the AP Newswire, as collected by AT & T Bell Laboratories, 1989
  - doe:
    short abstracts from the Department of Energy
  - fr:
    whole issues of the Federal Register, a publication that serves as a reporting source for actions taken by government agencies, 1989
  - wsj:
    stories from the Wall Street Journal, 1987-1989
  - ziff:
    information from the Computer Select disks, Ziff-Davis Publishing, 1989/1990

Statistics are given for the number of documents in the collection, the total size of the collection in number of terms in the vector representation of the database, the minimum, maximum and average lengths of the vectors representing the queries and documents, as well as the standard deviation of these vector lengths.

For more informations on the Virginia databases and the measurement method see [ASI96].

Table B.1 gives the number of distinct words (index terms) in the document dictionary, i.e. the dimensionality of query and document vectors.

Collection	*number of index terms*
CISI	6725
CRAN	5303
LISA	11739
Tipster	366911

Table B.1: Number of different terms in collection

Table B.2: QUERIES: *(All lengths in number of TERMS - After WEIGHTING)*

Collection	*Number of vectors*	*Total Size of the collection*	*Average Length*	*Standart Deviation of vector length*	*min length of a vector*	*max length of a vector*
CISI	112	3547	31.67	23.04	3	99
CRAN	225	2012	8.94	3.18	3	21
LISA	35	721	20.6	8.60	9	41
TIPSTER	200	9154	45.48	22.09	15	141

Table B.3: DOCUMENTS: *(All lengths in number of TERMS - After WEIGHTING)*

Collection	*Number of vectors*	*Total Size of the collection*	*Average Length*	*Standart Deviation of vector length*	*min length of a vector*	*max length of a vector*
CISI	1460	67228	46.05	19.18	9	165
CRAN	1400	78231	55.89	22.46	14	159
LISA	6004	208441	34.72	13.51	4	95
TIPSTER	510637	50033800	97.98	101.70	0	5144
ap	84678	12567424	148.41	73.73	1	617
doe	226087	10525673	46.56	19.15	0	143
fr	25960	4523479	174.25	196.76	0	5144
wsj	98732	13047617	132.15	115.78	1	1552
ziff	75180	9369607	124.63	130.18	3	3803

B.2 Vector Generation of the Tipster-collection

The vector representation of the Tipster collection had to be generated from the original plain text database. The plain text database contains a number of tags, some of which contain a manual classification of the documents contents. Different subcollections of Tipster use different tags.

These tags could give the clustering mechanism hints to which document sub-collection and to which topic a document belongs and should therefore be erased from the collection in order to get unbiased clustering results. The following list of tags is included to make the clustering results reproducible in different settings.

Text fields of the original database are marked by:

<tagname> text-field </tagname>

The contents of the text-field between a start-marker and end-marker of a text field denoted with a '0' are discarded, text-fields denoted with a '1' are included in the filtered database. Tags denoted with a '2' are ignored, i.e. if they occur in a text-field that is included in the filtered database, the contents between this markers is also included, if they occur in a discarded text-field, their content is also ignored (they are used mainly in the 'fr'-collection marking changes in the font style, e.g. bold-face or underlined characters).

All markers are removed from the database.

DOCNO	0	FILEID	0
TEXT	1	FIRST	0
SECOND	0	HEAD	1
DATELINE	0	BYLINE	1
NOTE	0	UNK	1
DOCID	0	SUMMARY	1
JOURNAL	0	TITLE	1
AUTHOR	1	DESCRIPT	1
ABSTRACT	1	COMPANY	0
CATEGORY	0	SPECS	0
PRODUCT	0	ADDRESS	0
FTAG	2	ITAG	2
T1	2	T2	2
T3	2	T4	2
T9	2	C	2
H1	2	H2	2
H3	2	H4	2
P	2	D	2
F	2	R	2
G	2	G7	2
HL	1	DD	0
SO	1	IN	1
LP	0	CO	0
GV	0	AN	0
RE	1	MS	0
NS	0	DOCID	0
DATE	0	DO	0
ST	0		

References

[ASI95a] A.M. AlHaj, E. Sumita, H. Iida: *A Parallel Text Retrieval System*, PPAI (IJCAI), 1995

[ASI95b] A.M. AlHaj, E. Sumita, H. Iida: *Massively Parallel Text Retrieval*, TR-IT-0114, ATR 1995

[ASI96] S. Auberger, E. Sumita, H. Iida: *A Comparative Study of Query Reformulation Methods on Vector Space Models*, TR-IT-0149, ATR 1996

[CKP92] D.R. Cutting, D.R. Karger, J.O. Pedersen, J.W. Tukey: *Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections*, SIGIR'92

[CKP93] D.R. Cutting, D.R. Karger, J.O. Pedersen: *Constant Interaction-Time Scatter/Gather Browsing of Very Large Document Collections*, SIGIR'93

[EGMS95] Efraimidis et al:*Parallel Text Retrieval on a High Performance Supercomputer Using the Vector Space Model*, SIGIR'95

[FrKu82] W. Fracis, H. Kucera: *Frequency Analysis of English Usage*, Houghton Mifflin, New York, 1982

[Hull94] D. Hull: *Improving Text Retrieval for the Routing Problem using Latent Semantic Indexing*, SIGIR'94

[Ide71] E. Ide, *New experiments in Relevance Feedback*, The SMART Retrieval System, ed. Salton, pp. 337-354, Prentice Hall, New Jersey 1971

[IwTo95] Iwayama Makoto, Tokunaga Takebonu:*Hierarchical Bayesian Clustering for Automatic Text Classification*, 95-TR0015, Tokyo Institute of Technology, 1995

[JaRi71] N. Jardine, C. van Rijsbergen:*The use of Hierarchic Clustering in Information Retrieval*, Information Storage & Retrieval 7, 1971

[Port80] C. Porter: *An Algorithm for Suffix Stripping*, Program 24(3), 1980

[RiCr75] C.J. van Rijsbergen, W.B. Croft: *Document Clustering*, Information Processing and Management 11, 1975

[SaBu88] G. Salton, C. Buckley: *Term Weighting approaches in automatic text retrieval*, Information Processing and Management 24, 1988

[SaBu90] G. Salton, C. Buckley: *Improving Retrieval Performance by Relevance Feedback*, Journal of American society for Information Science, 41(4), 1990

[Salt71] G.Salton, *The SMART Retrieval System*, Prentice Hall 1971

[StCa88] C. Stanfill, B. Kahle: *Parallel Free-Text Search on the Connection Machine*, Communications of the ACM, 29(12), 1988

[Voor85] E.M.Voorhees: *The Cluster Hypothesis Revisited*, TR85-658, Cornell University 1985

[Voor86] E.M.Voorhees: *The Effectiveness and Efficiency of Agglomerative Hierarchic Clustering in Document Retrieval*, Ph.D. thesis, Cornell University 1986

[Will88] P. Willett: *Recent Trends in Hierarchic Document Clustering: A Critical Review*, Information Processing & Management 24/5, pp 577-597