

Distributed Index Creation of Large Scale Web Collections in the Sidra System

Miguel Costa and Mário. J. Silva

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
1749-016 Lisboa, Portugal
{mcosta@xldb.fc.ul.pt, mjs@di.fc.ul.pt}

Abstract. Modern Web search engines have to prepare the future. Web collections keep growing and specialized algorithms are necessary to enable these systems to scale. Sidra is a new indexing and ranking system for large scale Web collections. It creates multiple distributed indexes organized and partitioned by different ranking criteria, aimed at supporting contextualized queries over hypertext data and metadata. This paper presents Sidra's architecture and the algorithms used to create its indexes. Results from the indexing of the Portuguese Web have shown that Sidra presents the scalability characteristics of global Web search engines.

1 Introduction

Information Retrieval (IR) systems use word indexes to speed up the search of relevant documents in a collection. Building these indexes for small collections of documents is a well studied and relatively easy to accomplish process. However, building a distributed index for large Web collections is a much more complex task. The Web continues to grow and modern Web search engines only index a small part of it. The deep Web, the part of the Web stored in online databases, is estimated to be 500 times larger than the known Web and still remains to be indexed [1].

Three factors make the development techniques for building large scale indexes a challenge. Fast indexing is a desirable feature, because Web search engines usually have time restrictions to update their indexes with fresher information. The system must be prepared to scale for the fast and unpredictable growth rate of Web collections, with the addition of inexpensive PCs. Specialized techniques are demanded for efficient indexing, overcoming the limitations imposed by the lack of memory and storage for collections of this magnitude.

Sidra is a new indexing and ranking system for large Web data sets that goes beyond keyword searching [2]. It maintains several distributed indexing data structures organized by different ranking criteria, which may be selected to find matches based on the context-data associated to queries. This approach significantly increases the storage requirements for indexes, but preserves the sub-second response times currently demanded by search engines' users.

Sidra was built for tumba! (see <http://www.tumba.pt>), a search engine for the Portuguese Web, which is being offered as a public service since 2002 [3]. Web search engines are composed by systems that crawl, archive, index and search information. Each one presents challenges difficult to overcome. This paper addresses the index creation system, presenting the architecture and algorithms implemented for the parallel and distributed generation of large scale Web indexes. In Section 2, we give some background on how a centralized indexing algorithm is usually implemented. In Section 3, we describe the architecture and the algorithms implemented in Sidra, while in Section 4 we present the results achieved during the indexing of a collection of documents delimited as the Portuguese Web. Section 5 explains why we opted for index reconstruction instead of supporting partial updates to indexes, and Section 6 presents the conclusion.

2 Centralized Indexing Algorithm

The indexes of large collections of documents can't be built in memory in a single step, given their size. First, the data of the index must be broken in smaller parts than the memory available. Then, each part is processed separately. An extensive analysis performed by Moffat, compares ten algorithms to build inverted indexes [4]. Some are unfeasible due to the large quantity of memory or storage needed, others because of the time spent in processing. As most of the time is spent in random disk seeks, algorithms with sequential access to files are the most efficient. Indexes are typically created in four phases:

1. Read and parse the documents. Each extracted token, called a term, are stored on a temporary file with associated information. Each term is associated on a triplet $\langle \text{term}, \text{docId}, \text{pos} \rangle$, with the identifier of the document, docId, and the position of the term in the document, pos. Other possible information may be added to each triplet, such as the font characteristics. Each of the triplets on the resulting file is called a hit.
2. Generate runs. As the hits of a large scale collection don't all fit in main memory, an external sort algorithm is used to sort them in batches, seeking to minimize the cost of disk accesses. Hits are divided in blocks of approximately the size of the available memory. These blocks are then sorted by term, id and position, in this order, using an in-memory sorting algorithm. Each of these sorted blocks stored on disk is called a run. The result is a set of runs.
3. Merge all runs (the next phase of the external sort). Runs are merged in pairs until only a single run with all the sorted hits remains. This can be accomplished by any external sorting program, such as the UNIX sort.
4. Generate an inverted file. In this indexed file, terms are the keys and the value of each key is the list of hits containing the term. To create it, hits are sequentially read from the run produced in the previous phase. Usually, the inverted file is compressed to reduce storage size and i/o transfer times.

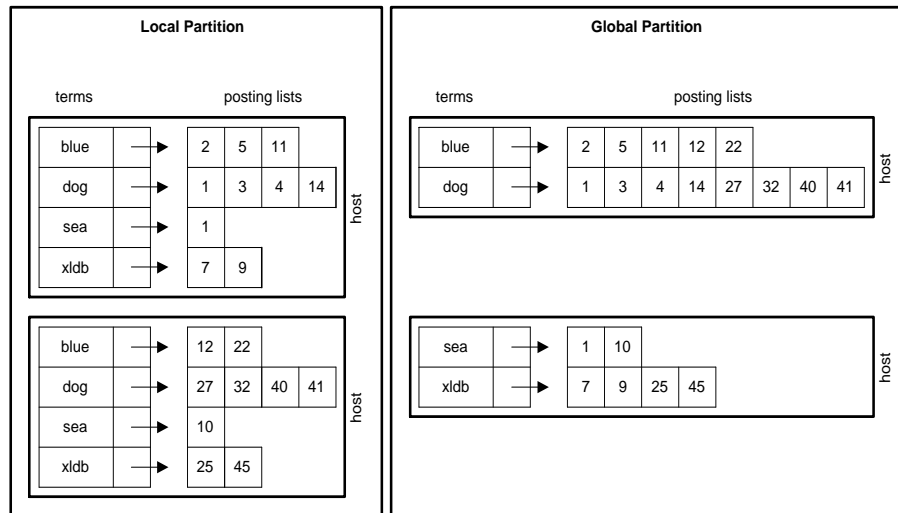


Fig. 1. Major proposals to partitionate inverted files.

3 Sidra's Distributed Indexing Algorithm

Unlike centralized indexes discussed above, Sidra can support multiple indexes, organized and partitioned by different criteria. Each index is used to search on a dimension of the data. Each partition indexes a subset of the data of a dimension.

Partitions can be local (also known as vertical) or global (also known as horizontal). Figure 1 depicts both organizations. In local partitioning, each computer has an inverted index of a disjoint set of documents of the collection. Each index partition is managed by a QueryServer, responsible for matching queries with the document identifiers indexed by the partition. The document identifiers received from each QueryServer are merged by a Broker and the top k returned to the user. In global partitioning, each computer has an inverted index with a disjoint set of terms of the collection. Brokers repeatedly request sets of document identifiers from the QueryServers indexing the query terms, and merge them until the top k ranked documents are obtained.

Sidra's indexes are globally partitioned by multiple QueryServers, allowing fast searches on different dimensions in parallel (partition parallelism). We have chosen a global partitioning scheme instead of a local partitioning, based on previous studies that have shown that it achieves better performance for bandwidths above 100 Mbps [5–7]. The explanation is that on a local partition all the computers have to respond to each query, while on a global partition only the computers with matching indexed terms reply.

To improve scalability of the creation of indexes of large Web collections, we developed a parallel distributed variation of the typical centralized indexing algorithm presented in the previous Section. This algorithm was designed to work

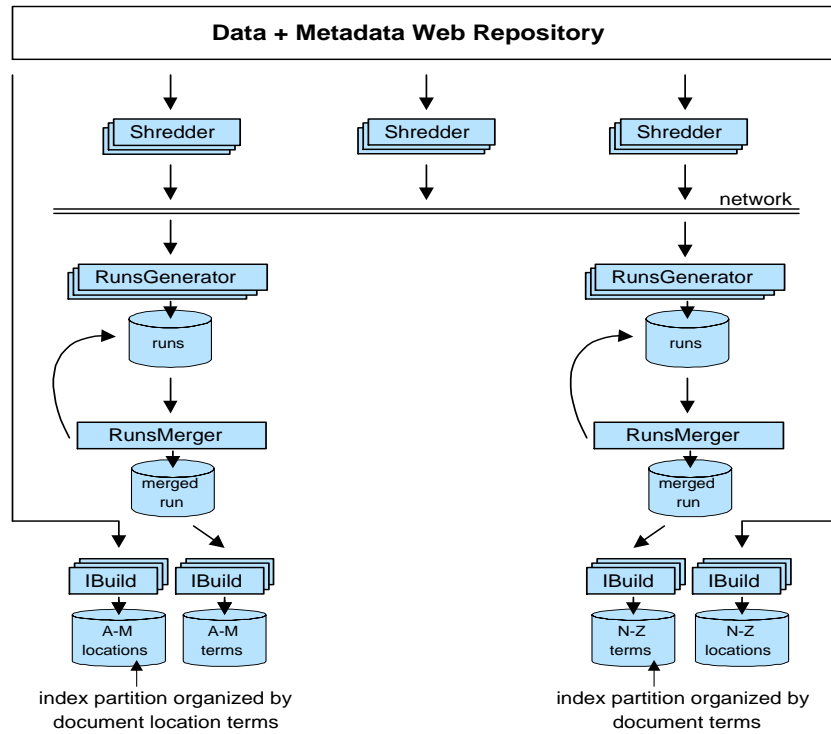


Fig. 2. Sidra's distributed architecture to create indexes.

over an environment of computer clusters, sharing nothing but the underlying network. Figure 2 depicts the architecture of the Sidra index generation system. The documents to index must have been previously crawled from the Web and stored in a Web repository. Indexing starts once the crawl is archived. The remainder of this section describes each of the phases of the distributed indexing algorithm.

3.1 Generating Runs

A multi-threaded program, the Shredder, parses documents and associated meta-data contained in the Web repository. In general, several Shredders run from a set of computers. The number and capacity of computers is defined based on the time that can be spent on processing the data to index. Shredders extract hits, identical to those of the first phase of the centralized indexing algorithm. Parsing the wide heterogeneity of Web pages is a complex task. The parser must be robust and tolerant against errors in Web pages. Sidra incorporates a Web data parser, WebCAT, also developed for tumba!, which handles a wide variety

of Web page formats, as HTML, MS Office formats, PDF (Portable Document Format), XML and more [8].

In Sidra, the parsed hits are not stored locally on disk. Instead, hits are directly sent to remote processes called RunsGenerators, that collect them to create index partitions. This way, we eliminate all i/o operations that would be required to write and then read the runs locally. Each RunsGenerator receives the hits assigned to its partition from all the Shredders. Then, sorts these hits with the quicksort internal sorting algorithm, and saves them into a run. To avoid network bottlenecks, Shredders send small messages containing hits instead of large batches, each time a buffer fills up. These messages have to be large enough to avoid penalties due to network overhead. This technique is derived from the RR algorithm, proposed by Ribeiro-Neto et al. [9].

In Sidra, we combine the RR technique with a technique presented by Melnik et al., based on software pipelines [10]. The creation of runs is partitioned in three distinct phases: loading, processing and flushing. During the loading phase, a number of pages are read from the network to memory. In the processing phase, hits are parsed and sorted using mostly CPU. In the flushing phase, the runs are stored on disk. These three phases are executed in pipeline and iteratively until no more pages remain to process. As each phase uses different resources, good concurrency is achieved through the parallelization of these three phases. There is optimal concurrency when all the resources are used simultaneously.

3.2 Merging Runs

At the end of the first phase, each RunsGenerator has received the hits necessary to create an index partition. The hits are organized and sorted in a set of files, the runs. On the second phase, the RunsMerger merges set of runs iteratively, until remains a single hits list file. We implemented several merge algorithms to identify which one offers the best response times. We chose a multiway merge algorithm with a replacement selection technique using a heap data structure [11]. The algorithm also makes use of a double buffering technique, used in database management systems to maintain the CPU busy during i/o requests [12]. Double buffering requires for each buffer of a run, another buffer of the same size. When no more hits remain to read from one buffer, they are read from the second buffer. At the same time, another thread fills up the empty buffer in parallel with hits read from the run file. This way, CPU idle times originated from i/o operations are eliminated. The same technique is applied to the output.

3.3 Building Inverted Files

At the start of this phase, all the hits of each partition are sorted in one single run. The creation of an inverted file is now a simple process. An index building program (IBuild) read the hits sequentially and creates a posting list for each distinct term.

Sidra, uses different indexes to search documents by different dimensions. Other IBuild applications paralelly read metadata associated to the documents

from the Web repository, and create partitions of index files corresponding to other index dimensions. In Figure 2 we show two possible partitioned indexes. One is the usual terms index. The other is a geographic index indicating the documents associated to terms representing locations composed by metadata from the Web repository.

In Sidra, posting lists are compressed with the Binary Interpolative Coding, a compression algorithm of integer posting lists, to minimize storage requirements and i/o latency [13]. This compression algorithm was chosen because it offers the best compression ratio and it is one of the fastest. After compressed, posting lists are stored on disk and became ready by the online query processing system (see [2]).

Some indexing systems use DBMSs to store inverted files [14–16]. This allows a faster development since transaction support is already available and there are tools to visualize and easily update the data. However, DBMSs don't enable a fine grained control over the data structures and core parts of the processing, necessary for a high performance system. Sidra, uses BerkeleyDB hash tables to manage and store indexes, an open source embedded database library that provides scalable, high-performance, and concurrent data management services to applications through an API [17]. BerkeleyDB provides a panoply of functionalities that reduce the time and complexity of a system like this, but at the same time, enables the optimization of the core parts of processing and storage. Sidra use hash tables because is the fastest way to access the posting lists. There is a very small probability of collision with the implemented FNV hashing function (see <http://www.isthe.com/chongo/tech/comp/fnv/>), but this is tolerable for a system of this nature.

4 Results

In this section, we present the performance and scalability results obtained with the existing implementation of Sidra indexing a collection of documents from the Portuguese Web, currently searchable with the tumba! search engine. We focused our analysis on the creation of the terms index partitions. Due to the large amount of data to process, this is the most computing intensive task in the index structures creation phase.

4.1 Testbed

Tests were performed on a cluster of 4 computers running RedHat 9 Linux with a 2.4.20 kernel, all connected by a 100 Mbps ethernet. Each computer has a 2.4 GHz Intel CPU, 1 GB of RAM and two mirrored disks. Each disk has a rotational speed of 7200rpm, an average seek time of 8.5 ms, and a media transfer rate of 699 Mbits/sec.

The Collection used is composed by 3,235,140 Web documents of different MIME types, with 78.4 Gigabytes of data (see [18], for a detailed characterization of this collection).

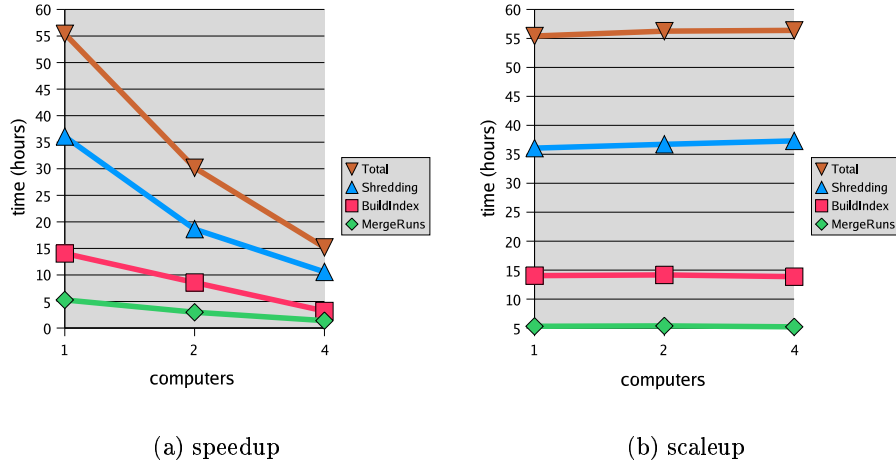


Fig. 3. Times to index the Portuguese Web varying (a) the number of computers, and (b) the number of computers and the collection size the same order of times.

During the first phase of the index creation, our test configuration of the system had one Shredder and one RunsGenerator component per computer. The next phases run independent in each of the available computers.

4.2 Tests and Analysis

Sidra indexed our collection of the Portuguese Web in 55.41 hours using a single computer. 20.2 hours of this time were spent retrieving and decompressing (with zlib) the documents from the Web repository. We also indexed the same collection with 2 and 4 computers. Run time decreased to 30.19 and 15.19 hours, respectively (see Figure 3(a)). These times show that Sidra exhibits nearly linear speedup in all the processing phases. This means that with p times more computers, the indexing time drops to $\frac{1}{p}$ of the total, which is essential to produce fresher indexes.

We also indexed the same collection replicated in the computers, simulating a collection 2 and 4 times larger. Times are almost constant, varying between 55.41 hours, when the collection is indexed with 1 computer, and 56.38 hours to index a collection 4 times larger with 4 times more computers (see Figure 3(b)). Sidra shows basically a linear scaleup. This means that, with p times more computers, Sidra can index a collection p times larger in the same time.

4.3 Comparative Results

Table 1 presents comparative results, showing Sidra’s performance data against similar published results. All these systems have different scopes and aims. Some

system	CPU #	collection			index		time (hr)	speed	
		size(GB)	type	decomp.	partition	comp.		1	2
Sidra	4	313.6	Web	yes	global	yes	56.38	1.39	16.2
RR	16	100	text	?	global	yes	6	1.04	?
Google	4	147.8	Web	yes	local	yes	147.4	0.25	13.5
CobWeb	312	500,000	Web	?	?	?	130.6	12.27	75

Table 1. Distributed algorithms to build inverted files. **collection decomp.** indicates if the documents need to be decompressed before indexed. **index comp.** indicates if the inverted index is compressed. **speed 1** is a measure of the number of Gigabytes indexed per CPU in one hour. **speed 2** is a measure of the number of pages indexed per CPU in one second.

indexed only textual data, while others parse Web documents. Some decompress documents before indexing, while others don't. The measures were obtained in different hardware configurations. However, despite the differences, this performance data can still be used as a baseline for comparison.

Ribeiro-Neto et al. developed three disk-based distributed algorithms to build global partitioned inverted files for large text collections [19]. The one that achieves the best results is the RR algorithm partially described in Section 3.1. After the computers receive the hits, they do multiway merges to produce a run and afterwards build an inverted file with it. Using 8 computers interconnected with a 80 Mbps network, the RR algorithm built a distributed inverted file for the 100 Gigabytes TREC-7 text collection in 12 hours. With twice the computers, it built the index in half of the time, demonstrating a perfect speedup.

Only a few articles describe indexing of large Web scale collections, all using local partitioning schemes. This is simpler than creating a distributed index with a global partition, because after distributing the documents by all computers, it isn't necessary to exchange further information, except if global statistics for ranking computation are necessary (e.g. inverse document frequency). The initial Google system created a forward index of 147.8 GB of data from a crawl of 24 million Web pages, at a speed of 54 pages per second [20]. This totalizes 123.4 hours. The transformation of this forward index into an inverted index took an additional 24 hours using 4 computers. We assumed that the same number of computers was used for the whole process.

CobWeb is the Internet Archive indexer and ranking system [21]. Their architecture is not published, only a few results are available from a slide show published in their Web site. It indexed more than 11 billion pages (0.5 PB of data) from the Internet Archive, the larger number known. A cluster of 312 computers with 512 MB of memory and 500 GB of disk each, has an indexing speed of 75 pages per computer in each second, totalizing 130.6 hours. The index has 2 Terabytes and grows sublinearly.

Table 1 shows that Sidra has performance results comparable to other systems, when the size of the indexed collection is on the order of 100-300 GB and the number of processors until 16. CobWeb operates on a completely differ-

ent range: one order of magnitude more processors, a collection three orders of magnitude larger, and an indexing speed one order of magnitude faster. However, the algorithms and software architecture of this indexing system are not publicly documented. Sidra provides speedup and scaleup characteristics which enables it to built indexes for very large Web collections as fast as necessary, by only adding more computers to the processing. This motivates the following discussion of the need to support index updates on Web search engines.

5 Index Updates

Since Web contents change continuously, there is a need to refresh the indexes of Web search engines frequently. Cho and Garcia-Molina analyzed the evolution of the Web during 4 months and discovered that 23% of the pages change every day, 15% change between a day and a week, and 16% change between a week and a month [22]. Fetterly et al. extended their work and presented results derived from eleven weeks of analysis [23]. During this time, 34.8% of the pages changed something, but the text of the pages only changed on around 10% of the pages. Given these conditions, it seems that an indexing system could profit from performing partial updates to indexes, instead of full reconstruction.

However, partial updates have their own problems: (1) the complexity of the indexing system increases, since indexes have to respond to queries, while being updated with new versions of pages at the same time; (2) index sizes increase due to internal fragmentation in the posting lists. Most techniques to update indexes, reserve space apriori in the posting lists to add new postings in the future [24, 25]. In the majority of the systems, this also causes a degradation of query performance, because partial updates of indexes link existing posting lists with new ones on different disk blocks. As postings lists are not continuous in disk, i/o and consequently the overall time increases; (3) for an efficient update, it is necessary a forward index of the collection. This has sensibly the same space of the inverted file; (4) beyond all that and contrary to the expected, Cho and Garcia-Molina showed that for crawls done periodically each month, partial updates provide indexes with freshness similar in average to the ones completely rebuilt [22].

Given this knowledge of previous Web studies and the performance observed with Sidra, it does not seem profitable to support partial index updates. Sidra enables fast creation of indexes for specific collections with pages changing at a higher frequency. Sidra can also support queries to multiple collections on the same computers. This functionality enables tumba! to serve specialized indexes. This is also the path followed by some Web search engines. Google for instance, crawls and index from scratch around 3 billion pages about once a month (see <http://www.google.com/webmasters/2.html>), and offers a specialized search engine for news, updated daily (see <http://news.google.com>). In the future we will implement a mechanism to search in parallel indexes created with different periodicities. Only the fresher versions of the documents indexed will be ranked and presented to the user.

6 Conclusion

Web collections are growing at a fast rate. Indexing systems, which are the backbone of Web search engines, need an efficient architecture and algorithms with scalability characteristics to keep up with this growth. Sidra is a new indexing and ranking system for large scale Web collections, enabling the creation of several indexes that contextualize queries to different searchable dimensions. Sidra distributes the indexes using global partition, which presents the best high-performance results for query processing. Sidra combined several algorithms and techniques applied on traditional IR systems, databases and Web search engines, to produce a high performance system capable of scaling the index creation of large scale Web collections of documents. These properties were validated with the indexing of the Portuguese Web, currently searchable on the tumba! Web search engine.

References

1. Bergman, M.K.: The deep Web: surfacing hidden value. *The Journal of Electronic Publishing from the University of Michigan* **7** (2001)
2. Costa, M., Silva, M.J.: Sidra: a flexible distributed indexing and ranking architecture for Web search. In: *Proceedings of the VIII Conference on Software Engineering and Databases JISBD 2003*. (2003)
3. Silva, M.J.: The case for a Portuguese Web search engine. In: *Proceedings of the IADIS WWW/Internet 2003 Conference*. (2003)
4. Moffat, A.: Resource-limited index construction for large texts. In: *Proceedings of the 17th Australasian Computer Science Conference*. (1994) 169–178
5. Tomasic, A., Garcia-Molina, H.: Performance of inverted indices in distributed text document retrieval systems. In: *Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems*. (1993) 8–17
6. Jeong, B.S., Omiecinski, E.: Inverted file partitioning schemes in multiple disk systems. *IEEE Transactions on Parallel and Distributed Systems* **6** (1995) 142–153
7. Ribeiro-Neto, B., Barbosa, R.A.: Query performance for tightly coupled distributed digital libraries. In: *Proceedings of the 3rd ACM Conference on Digital libraries*. (1998) 182–190
8. Martins, B., Silva, M.J.: WebCAT: A Web content analysis tool for IR applications. To be submitted (2004)
9. Ribeiro-Neto, B., Moura, E.S., Neubert, M.S., Ziviani, N.: Efficient distributed algorithms to build inverted files. In: *Proceedings of the 22th International ACM SIGIR Conference on Research and Development in Information Retrieval*. (1999) 105–112
10. Melnik, S., Raghavan, S., Yang, B., Garcia-Molina, H.: Building a distributed full-text index for the Web. In: *World Wide Web*. (2001) 396–406
11. Sedgewick, R.: *Algorithms in C*. Addison Wesley (1990)
12. Ramakrishnan, R.: *Database Management Systems*. McGraw-Hill Computer Science Series (1998)
13. Moffat, A., Stuiver, L.: Binary interpolative coding for effective index compression. *Information Retrieval* **3** (2000) 25–47

14. Brown, E.W., Callan, J.P., Croft, W.B., Moss, J.E.B.: Supporting full-text information retrieval with a persistent object store. In: Proceedings of the 4th International Conference on Extending Database Technology—EDBT'94. (1994) 365–378
15. Frieder, O., Chowdhury, A., Grossman, D., McCabe, M.C.: On the integration of structured data and text: A review of the SIRE architecture. In: DELOS Workshop on Information Seeking, Searching and Querying in Digital Libraries. (2000)
16. Grabs, T., Böhm, K., Schek, H.J.: PowerDB-IR: information retrieval on top of a cluster of databases. In: Proceedings of the 10th International Conference on Information and Knowledge Management. (2001)
17. Olson, M., Bostic, K., Seltzer, M.: Berkeley DB. In: Proceedings of USENIX Technical Conference, FREENIX Track. (1999)
18. Gomes, D., Silva, M.J.: A characterization of the Portuguese Web. In: Proceedings of 3rd ECDL Workshop on Web Archives. (2003)
19. Ribeiro-Neto, B., Kitajima, J.P., Navarro, G., Sant'Ana, C., Ziviani, N.: Parallel generation of inverted files for distributed text collections. In: Proceedings of the 18th International Conference of the Chilean Computer Science Society. (1998)
20. Brin, S., Page, L.: The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems* **30** (1998) 107–117
21. Patterson, A.: Cobweb search. <http://ia00406.archive.org/cobwebsearch.ppt> (2004)
22. Cho, J., Garcia-Molina, H.: The evolution of the Web and implications for an incremental crawler. In: Proceedings of the 26th International Conference on Very Large Databases. (2000) 200–209
23. Fetterly, D., Manasse, M., Najork, M., Wiener, J.L.: A large-scale study of the evolution of Web pages. In: Proceedings of the 12th International World Wide Web Conference, WWW2003. (2003) 669–678
24. Brown, E.W., Callan, J.P., Croft, W.B.: Fast incremental indexing for full-text information retrieval. In: Proceedings of the 20th International Conference on Very Large Databases (VLDB). (1994) 192 – 202
25. Tomasic, A., Garcia-Molina, H., Shoens, K.A.: Incremental updates of inverted lists for text document retrieval. In: Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data. (1994) 289–300