Toward a Multi-Tier Index for Information Retrieval System

EMAD S. ELABD, AHMED Z. EMAM ,NABIL A. ISMAIL,FAWZY A. TURKY
Information System Department and Computer Science Department
Menoufia University, Faculty of Computers and Information
Shebin Elkoom , Menoufia
EGYPT
http://mufic.com

Abstract: - Text Information Retrieval(TIR) is considered the heart of many applications such as Document Management System(DMS). TIR that used for DMS requires different techniques of data structure than that used in the search engine. Search engine, requires special hardware (super computers with high memory) to perform information retrieval algorithms. In this paper, a new approach is developed to make it easy for DMS to perform the retrieval process with high performance. Conventional approaches are based on single inverted file but our approach is based on object and multi-tier inverted index files structure.

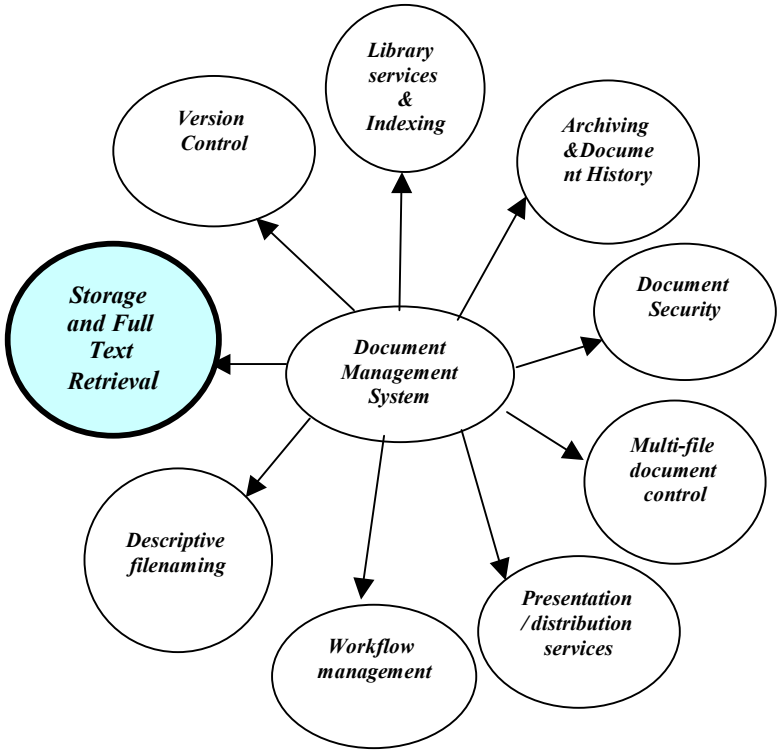
Keywords: Text Information Retrieval (TIR) , Inverted Index file, Multi-tier Inverted file, Document Management Systems (DMS)

1- Introduction

Information is the most important resource for all organizations either large or small and considered as valuable assets. The amount of information to be dealt with has increased dramatically with the advent of Internet and other communication means. The growing need to manage information abundance has given rise to fast evolution of Knowledge Management Technology (KMT). KMT core and fundamental component is Document Management Systems (DMS). The information overload problems in information intensive organizations may be way out by technologies providing intelligent information retrieval with easy access to information from anywhere to anyone[1].

The DMS are relatively new so the definition is fluctuated and not concrete and in many cases the definition of DMS is application dependent. The main functions of the document management system different from one application to another. For example, the DMS used in law company concentrate on the version functionality of the application and the DMS used in financial organization concentrate on the security functionality. Therefore, DMS can be defined as a class of information technology that is used to coordinate electronic document management, storage, and retrieval [2].

Another definition stated that document management is the automated control of electronic documents through their entire life cycle, from creation to archiving. The DMS basic functions are Storage and Full Text Retrieval, Descriptive File Naming, Multi-file Document Control, Version Control, Archiving, Library Services and Indexing, Document Security, Workflow Management, Document History, and Presentation/Distribution Services as shown in fig.(1).



Figure(1): Basic functions of Document Management System

Storage and Full Text Retrieval is considered to be one of the most important functions of DMS. Many researches developed different approaches tends to improve the performance of the text retrieval process.

1-1 IR indexing techniques

The first phase of Text Retrieval (TR) process is indexing process. There are three main categories of TR indexing techniques : *inverted files , suffix arrays and suffix trees, and signature files*. Inverted files technique is considered to be currently the best choice for most applications. The main advantages of inverted files technique is easy to build and Keyword-based search. Suffix trees and arrays are faster for phrase searches and other less common queries, but are harder to build and maintain. Finally, signature files were popular in the 1980s, but nowadays inverted files outperform them[6]. From the view of query evaluation speed and space requirements, inverted files are distinctly superior to signature files. Also, inverted files require less space and provide greater functionality than signature files. So, the nominated techniques for our work is inverted files. There are many effected factors that affect the performance of inverted files approach. Construction time of the inverted files and searching time using it are the critical factors in text retrieval process.

The paper organized as follows : section 2 describes the previous work , section 3 defines the problem statement, section 4 describes the proposed approach , section 5 contains the experimental results, and the last section is the summary and conclusion.

2- Literature Review

The previous work in this research field follows mainly two directions. First direction is sequential processing, which use one processor at a time to construct the inverted files index used in the information retrieval. Second direction is the parallel processing, which uses multi processor to construct the inverted file index. Both cases produce one file indexes all the documents or multi files have the same format each file indexed some documents.

2-1 Sequential processing

An inverted file is the sorted list of keywords (attributes), with each keyword having links to the documents containing that keyword. The first file that contains list of keywords is called dictionary file and the second file that contain the documents linkage is called the posting file, as shown in fig(2). Harman[3] mentioned that there are many structures that can be used to implement inverted files: *sorted arrays, B-trees, tries, and various hashing structure*, *or hybrid structures*. This inverted file can be searched using binary search. So, using sorted arrays can be easy to implement and are reasonably fast for search.

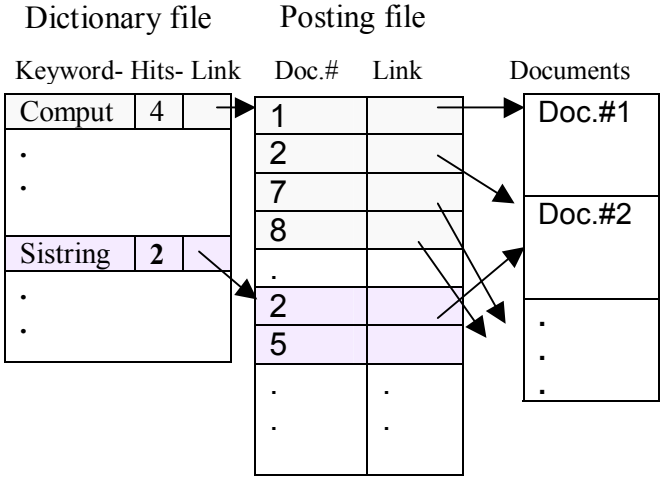


Figure (2): Implementation of inverted file using sorted array.

Yates [6] described an approach called the full inverted indices. That approach used a full position description for each word in the document and occurrences number. The shortcoming from using this approach is the overhead size of the inverted file, which represents 30% to 40% of the original document.

M. Marin [17] stated that the size of inverted files is large and take space, which is 30% to 100% of the original document size.

a system called Glimpse is implemented using block addressing idea to speedup the construction of the inverted file is developed in [4]. Later, block addressing indices are analyzed by R. Beaza-Yates and G.Navarro.[5], where some performance improvements are proposed. The main advantages of using block addressing is the shrinking of the inverted file size to become only 5% overhead of the original text size.

Yates [6] provided a complete comparison among four types addressing used in indexing process. Table (1) shows the sizes of the inverted file as approximate percentages of the size the of whole text collection. Four granularities and three collections are considered in this comparisons.

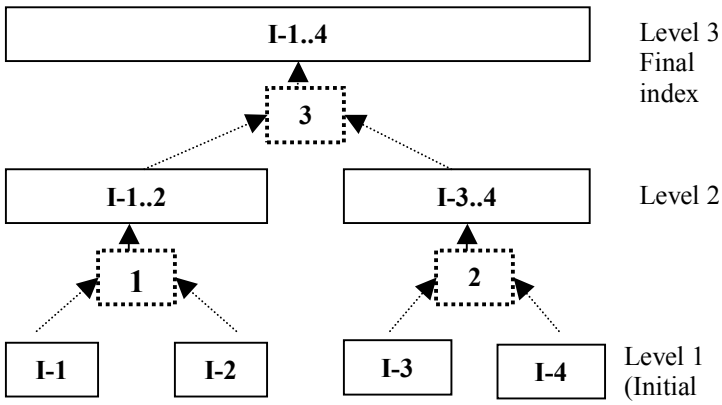
Table (1): sizes of the inverted file as approximate percentages of the size the whole text collection

	Indexing Type	Small collection (1 MB)		Medium collection (200MB)		Large Collection (2GB)	
		All But Stop Word	All words	All But Stop Word	All words	All But Stop Word	All words
*	Addressing words	45%	73%	36%	64%	35%	63%
**	Addressing documents	19%	26%	18%	32%	26%	47%
***	Addressing 64k blocks	27%	41%	18%	32%	5%	9%
***	Addressing 256 blocks	18%	25%	1.7%	2.4%	0.5%	0.7%

Baeza-Yates and B.Ribeiro-Neto, Modern Information Retrieval, page 195

* Full inversion (all words,.exact positions, 4-byte pointers)
** Document size (10KB), 1, 2,3 bytes per pointer, depending on text size
*** 2 or 1 byte(s) per pointer independent of the text size

The inverted files construction process entirely performed into the main memory and the resultant files stored eventually into secondary storage disk. Building inverted files in memory requires very high memory size, which may or may not be available. For large text files, the inverted file will not fit into the memory, which force to utilize the swapping/partial indexing techniques. Yates[6] presented the partial index approach by dividing the original text file into small buckets, which appropriate to fit into the main memory. Then merging all partial indices resulted in the same level in a hierarchical manner. Finally store the final resulted index file into the secondary storage disk. The whole process described in fig.(3).



Figure(3): Partial Indices technique merging the partial indices in a binary fashion. Continuos rectangular represent partial indices, while dashed one represent merging operations . the numbers inside the merging operation show a possible merging order

Harman and Candela (1990) developed a new method of producing an inverted file for large data sets without Sorting. The main idea was to avoid using of explicit sorts by right-threaded Binary tree.

A Fast Inversion Algorithm is another technique called FAST-INV developed by Fox and Lee in Virginia Tech. This technique assumes two principles. The first principle is the large primary memories are available to support large size databases indexing and reduce the overall cost. The second principle is the inherent order of the input data to avoid very expensive polynomial or even $n\log_n$ sorting algorithms complexity for large files.

2-2 Parallel processing

Bulk-Synchronous Parallel (BSP) model of computing had been proposed to enable the development of portable and cost-predictable software which achieves scalable performance across diverse parallel architectures[14, 15]. BSP is a distributed memory model with a well-defined structure that enables the prediction of running time. Unlike traditional models of parallel computing, the BSP model ensures portability at the very fundamental level by allowing algorithm design to be effected in a manner that is

independent of the architecture of the parallel computer. Shared and distributed memory parallel computers are programmed in the same way as they are considered emulators of the more general BSP machine[17].

Inverted index files construction and utilization by BSP model has been tackled using two approaches, which are local index approach and global index approach[9, 10, 11, 12, 13]. In the local index approach the documents are assumed to be uniformly distributed onto the processors. A local inverted-lists index is constructed in each processor by considering only the documents there stored respectively. For example, assume a server operating upon a set of *p* identical machines, each containing its own main and secondary memory, then *p* individual inverted-lists structures performed. If a query consisting of one term must be solved by simultaneously computing the local sub-lists of document identifiers in each processor then producing the final global list from these *p* local sub-lists. The second approach utilize the whole collection of documents to produce a single inverted lists index, which is identical to the sequential one.

Distributed algorithms for building global inverted files for large collections distributed across a high-bandwidth network of workstations were discussed by Ziviani et al.[16]. Three distributed algorithms provided to build global inverted files for very large text collections. The distributed environment they use is a high bandwidth network of workstations with a shared-nothing memory organization. The text collection assumed to be evenly distributed among the disks of the various workstations. These three algorithms are Local Buffer and Local Lists (LL Algorithm), Local Buffer and Remote Lists (LR Algorithm), and Remote Buffer and Remote Lists (RR Algorithm). These algorithms differ in the place to build the local and global inverted files

In summery, there are two directions to construct inverted files used in IR; sequential and parallel direction. Both directions require unlimited memory for reasonable results.

3- Problem Statement

The process of searching data in DMS depends on the used IR techniques. The most common indexing technique uses the inverted index file, which represent data as indexed data. Literature review in the pervious section point out that traditional method for construction inverted file depends on very high specification computers systems, which may not be available for small companies. This was the most important barrier for small companies to adapt DMS. In addition, searching a data into inverted file and update the inverted file are the main processes for information retrieval into DMS. The critical affected factors of indexing process are the construction time and the searching time. The common trend tends to reduce the construction time using parallel concept, which requires more hardware. The proposed approach based upon using multi-tire partial indexing to reduce the construction time for inverted index file and use less hardware requirement to perform faster search.

4- Proposed Approach

The inverted index file constructed from the developed algorithms is one file with each record of the file contains (word, number of documents containing the word ,total frequency , [id-frequency] of each document the word appeared in) as shown in fig.(4)

Word	*number of documents contain the word*	*total frequency*	*[id-frequency] of each document the word appeared in*
Ram	3	6	*1-3*(ram is appeared in DOC 1 three times) *2-1* *3-2*

Figure (4): record of the inverted file used in the previous methods .

The second method for construction is the inverted index file which represents two associated files, the first file (dictionary) contains (word, number of documents containing the word ,total frequency, pointer to the second file) and the second file (posting) contains ([id-frequency] of each document the word appeared in it) as shown in fig.(5)

Dictionary			**posting**	
Word	Number of documents contain the word	total frequency	id	frequency
Ram	3	6	1 2 3	3 1 2

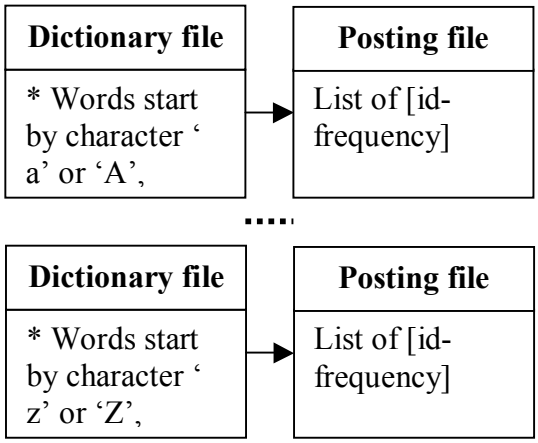
Figure (5): record of the inverted file used dictionary files and posting files

In search of a word in the inverted index file, we search of the word on the whole file and get the record of it, which contains all information related to that word.

There are several experiment done before developing the proposed technique. The first trial was building inverted index file as a tree structure for all the text in the main memory. In case of small computer systems, the large text files led to large tree size, which consume the whole memory and reduce the system performance.

The next direction was to utilize the fixed size sorted array structure. That structure enhance the performance but the whole text size was undetermined, which make it hard to shape out the size of that array. Therefore, the dynamic sorted array with partial indices concept was the nominated direction to improve the system performance. This method is considered as one of the reliable and appropriate approaches in construction of the inverted index file. thus, this approach will be the basic and reference structure technique for the new approach. In the pervious developed methods, quick sort technique was used as sorting technique for all sorted array and binary search used as search technique.

The new approach based on partial indices and multi-tier concept by creating a separate files for all words starts by the same alphabetic character and one more file for all words start by special or numeric characters as a second tier. The directory of the twenty seven files considered as a first tier used in searching process. The number of resulted inverted files is 27 file and each record of that file consist of (word starts by that character, number of documents contain the word, total frequency, [id-frequency] of each document the word appeared in it). Another method is constructing 27 files as dictionary files and 27 as posting files. The second method is preferable to perform better in search and memory space. The construction process for each file is shown in fig. (6) and real example presented in fig. (7).



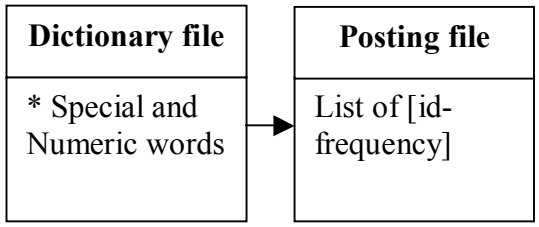


Figure (6) : inverted files created in our approach

Dictionary			Posting	
word	number of documents contain the word	total frequency	id	frequency
amber	3	6	1 2 3	3 1 2

Figure (7): Example for a record of inverted files created for character 'a'.

The main benefits for using multi-tier design will be enlarged search process for any query. The first step is looking up in the first tier file (file director) to identify the first letter in the query and determine the file name in the second tier to perform the search on it. The second step is search in second tier, which is directed and accurate by loading the directory and posting files into the main memory.

5- Experimental Results

Real dataset considered as valuable assets for any organization, the easy and inexpensive cost is synthetic dataset creation. Therefore, all experiments used in this research is synthetic dataset. The creation of synthetic dataset was done using a random function generator for words to create a text document.

The partial indices as conventional approach for constructing inverted index file was implemented using a Visual Basic Ver6.0 on windows platform using two different hardware systems. The first hardware system is PII333 MHZ with 64MB RAM. The second hardware system is Dell server 2.8GHZ with 1GB RAM.

Table (2) columns represent the size of original text file, size of index file, construction time, and worst average time for searching in case of using Dell server with the pervious configuration and perform partial indexing technique.

Table (2) : Inverted files construction and searching time using Partial Indexing using Dell Server

Partial Indexing Technique			
Text File Size	Size of inverted file	Construction Time /s	Searching Time /s
1K	2K	0.094	0.016
4K	4.5K	0.109	0.016
8K	5K	0.125	0.016
16K	14.5K	0.156	0.016
32K	17K	0.219	0.016
64K	56K	0.313	0.016
128K	67K	0.656	0.016
256K	89K	0.953	0.016
512K	134K	1.782	0.016
1M	2.72M	10.906	0.281
2M	3.27M	23.297	0.297
4M	4.3M	49.469	0.313
8M	5.2M	106.031	0.313
10M	5.43M	123.062	0.578
20M	6.51M	261.828	0.578
30M	7.49M	397.766	0.578
50M	9.48M	714.563	0.593
100M	14.6M	1553.797	0.61

Table (3) : Inverted files construction and searching time using Multi-tier Indexing using Dell Server

Multi-Tier Indexing Technique			
Size of text file	Size of inverted file	Construction Time /s	Searching Time /s
1K	20.5K	1.225	0.016
4K	26.5K	1.235	0.016
8K	26.5K	1.250	0.016
16K	27.5K	1.312	0.016
32K	40K	1.328	0.016
64K	62K	1.469	0.016
128K	76K	1.609	0.016
256K	99.5K	1.985	0.016
512K	142K	2.469	0.016
1M	2.57M	10.125	0.016
2M	3.08M	21.172	0.016
4M	4.14M	45.219	0.016
8M	6.18M	97.921	0.016
10M	5.05M	99.750	0.031
20M	6.2M	220.140	0.031
30M	7.21M	358.094	0.031
50M	9.17M	647.953	0.031
100M	14.2m	1407.687	0.032

Table (3) columns represent the size of original text file, size of index file, construction time, and worst average time for searching in case of using Dell server with the pervious configuration and perform multi-tier indexing technique.

The inverted index file size and the original text size are relatively correlated as shown in figures (8) and (9) using Dell Server and *PII 333 Mhz machines respectively*, which means that partial indexing and multi-tier indexing techniques almost have the same effect in the creation process of index file. Construction time of the inverted index file is one of the most important factors to measure the IR system performance. Fig.(10) shows that there is no significant difference in construction time using partial index technique or multi-tier index technique on Dell server. While, fig.(12) shows a major and highly significant difference in searching time (worst case average) using multi-tier index technique than using partial index technique on Dell server. On other words, multi-tier indexing techniques have superior performance than partial index technique on Dell server platform.

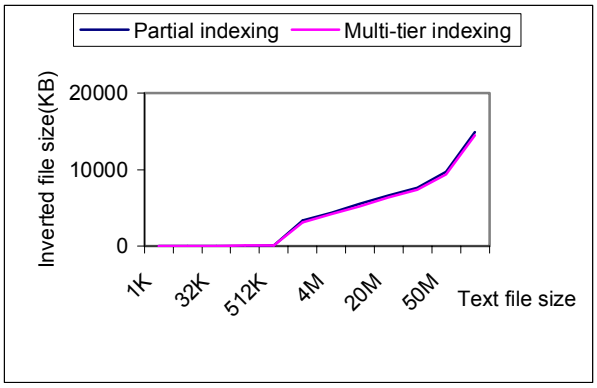


Figure (8):Inverted files size using Partial Indexing and multi-tier on Dell Server

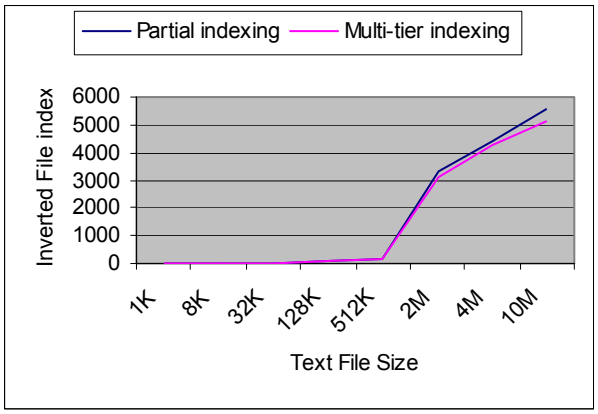


Figure (9):Inverted files size using Partial Indexing and multi-tier on PII 333 Mhz

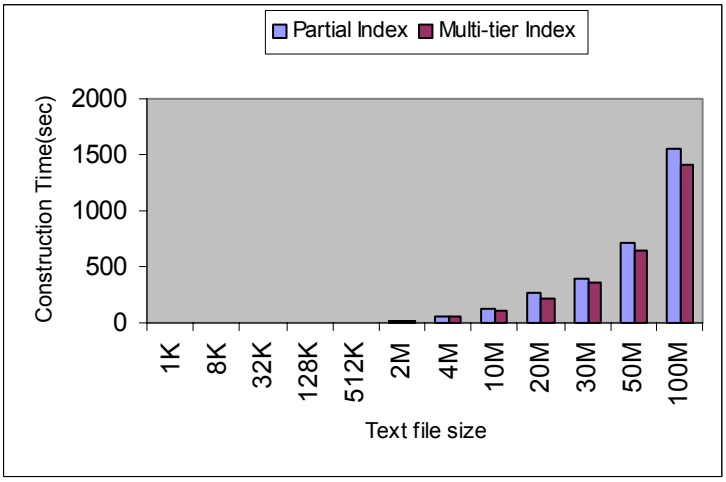


Figure (10): Inverted files construction time using Partial Indexing and multi-tier on Dell Server

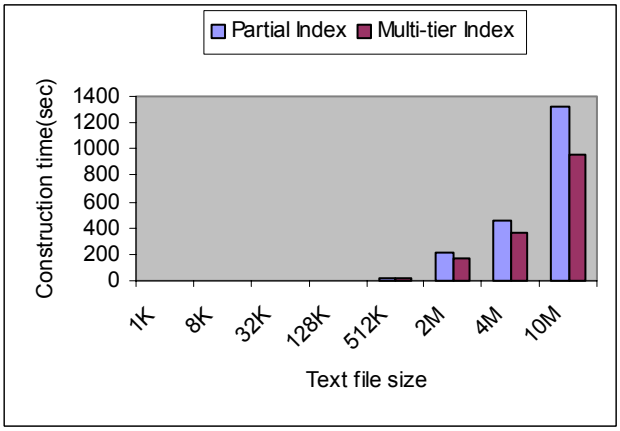


Figure (11): Inverted files construction time using Partial Indexing and multi-tier on PII 333 Mhz

Table (4) : Inverted files construction and searching time using Partial Indexing using PII 333Mhz

Partial Indexing Technique			
Text File Size	Inverted File Size	Construction Time /s	Searching Time /s
1K	839byte	0.39	0.060
4k	3.34K	0.49	0.060
8K	4.13K	0.55	0.060
16K	13.5K	0.82	0.060
32K	16.5K	1.21	0.060
64K	55.1K	2.14	0.060
128K	66.4K	3.84	0.060
256K	88.5K	7.25	0.060
512K	133K	14.17	0.060
1M	2.27M	97.99	2.690
2M	3.27M	211.41	2.690
4M	4.3M	460.93	2.750
8M	6.32M	1020.18	2.750
10M	5.43M	1323.93	5.220
20M	N/A	N/A	N/A
30M	N/A	N/A	N/A
50M	N/A	N/A	N/A
100M	N/A	N/A	N/A

Table (5) : Inverted files construction and searching time using Multi-tier Indexing using PII 333 Mhz

Multi-Tier Indexing Technique			
Text File Size	Inverted File Size	Construction Time /s	Searching Time /s
1K	0.86K	3.02	0.050
4k	3.11k	2.58	0.050
8K	3.9K	2.69	0.050
16K	12.9K	3.35	0.050
32K	15.5K	3.57	0.050
64K	50.9K	3.79	0.050
128K	62.9K	4.67	0.050
256K	85.1K	6.81	0.050
512K	128K	10.82	0.050
1M	2.56M	74.09	0.110
2M	3.06M	162.58	0.160
4M	4.13M	360.75	0.110
8M	6.17M	815.09	0.110
10M	5.04M	955.54	0.220
20M	N/A	N/A	N/A
30M	N/A	N/A	N/A
50M	N/A	N/A	N/A
100M	N/A	N/A	N/A

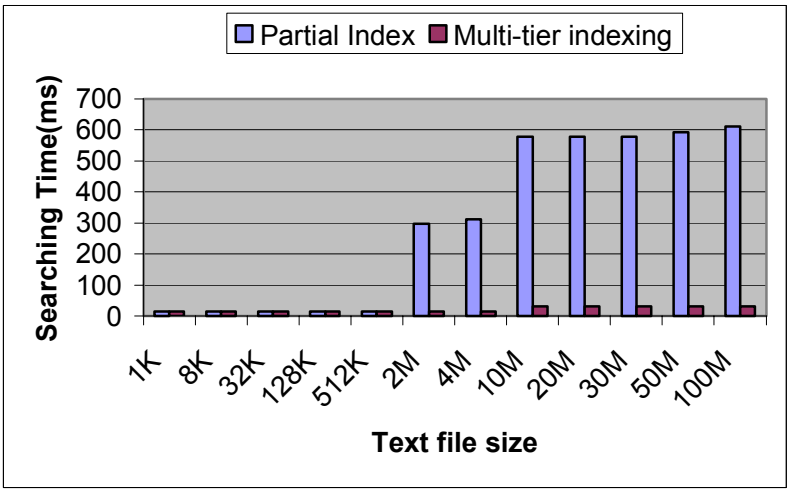


Figure (12): Inverted files searching time using Partial Indexing and multi-tier on on Dell Server

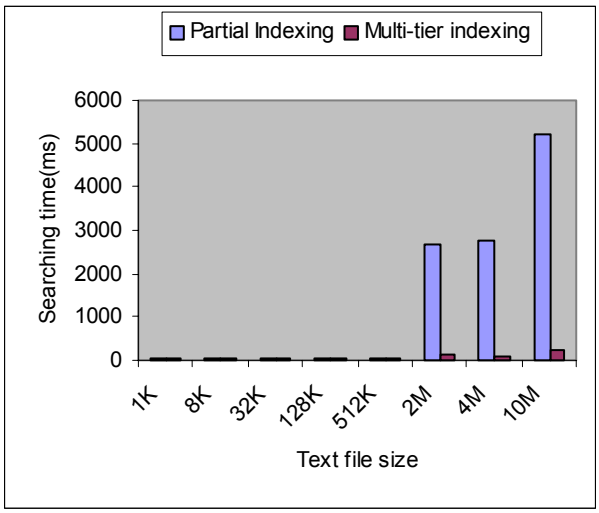


Figure (13): Inverted files searching time using Multi-tier Indexing PII 333Mhz

Like Dell server platform, the index file size and the original text size are relatively correlated as shown in tables (4) and (5) when using PII 333 Mhz. platform. This was an indictor to both partial indexing and multi-tier indexing techniques which create an index file with the same size. On other words, the size of index file from both techniques are similar. fig.(11) shows that there is no significant difference in construction time using partial index technique or multi-tier index technique on PII 333 Mhz as on Dell server. While, fig. (13) proof that there is a big significant difference in searching time (worst case average) using multi-tier index technique than using partial index technique on PII 333 Mhz.

6 – Conclusion and Future Work

It has been shown that Partial indexing technique and multi-tier indexing technique have the same or similar effect in construction time and index file size. The index file searching for a query is the most important factor in IR performance evaluation. The proposed approach proofs that the searching time using multi-tier index is almost a linear complexity and performance that can be predicated. On other words, whatever the size of the original text file the searching performance using a multi-tier index technique is relatively constant.

There are several modifications that can be applied in the proposed approach such as increasing the level of multi-tier index technique instead of two levels. Parallelizing the construction process using multi-threads techniques may also be applied. The role of different operating systems is considered to be one of the facts that should be studied in our approach.

References:

[1] J. Vain and Juhan-Peep Ernits , “Electronic Document Management” ,” http://www.cc.ioc.ee/training/unesco/onlinegov /docsys/detailed/, UNESCO project "Developing Telematics and Information Networks for On-Line Governance" at the Tallinn Technical University

[2] G. Cleveland, Document Management Systems, Information Technology Services, National Library of Canada ,March 7, 1997 .

[3] D. Harman, E. Fox, W. Lee and R. Baeza-Yates. Inverted files. In W. Francks and R. Baeza-Yates ,editors, Information Retrieval: Datastructure & Algorithms ,Chapter 3, pages 28-43. Prentice Hall, England Cliffs, NJ, USA,1992

[4]Udi Manber and Sun Wu. GLIMPSE:A tool to search through entire file systems. In Proc. Of USENIX Technical Conference , pages 23-32 , San Francisco,USA,January 1994. ftp//cs.arizona.edu/glimpse.ps.z.

[5] R. Baeza-Yates and G. Navarro. Block-addressing indices for approximate texte retrieval. In Proc. Of the 6th CIKM Conference , pages1-8, Las Vegas, Nevada, 1997.

[6] R. Baeza-Yates and B.Ribeiro-Neto,Modern Information Retrieval, Chapter 3, Indexing and Searching , with Gonzalo Navarro. New York: Addison Wesley, 1999.

[7] J. ZOBEL , A. MOFFAT and K. RAMAMOHANARAO : Inverted Files Versus Signature Files for Text Indexing. Draft, June 19, 2002

[8] M. Araujo, G. Navarro, and N. Ziviani. Large text searching allowin errors . In Proc. WSP’97 , pages 2-20 , Valparaiso, Chile, 1997. Carleton University Press .

[9] C. Badue, R. Baeza-Yates, B. Ribeiro, and N. Ziviani. Distributed query processing using partitioned inverted files. In Eighth Symposium on String Processing and Information Retrieval (SPIRE'01), pages 10-20. (IEEE CS Press), Nov. 2001.

[10] S.H. Chung, H.C. Kwon, K.R. Ryu, H.K. Jang, J.H. Kim, and C.A. Choi. Parallel information retrieval on a sci-based pc-now. In Workshop on Personal Computers based Networks of Workstations (PC-NOW 2000). (Springer-Verlag), May 2000.

[11] A.A. MacFarlane, J.A. McCann, and S.E. Robertson. Parallel search using partitionedinverted files. In 7th International Symposium on String Processing and Information Retrieval, pages 209-220. (IEEE CS Press), 2000.

[12] B.A. Ribeiro-Neto and R.A. Barbosa. Query performance for tightly coupled distributed digital libraries. In Third ACM Conference on Digital Libraries, pages 182-190. (ACM Press), 1998.

[13] A. Tomasic and H. Garcia-Molina. Performance of inverted indices in sharednothing distributed text document information retrieval systems. In Second International Conference on Parallel and Distributed Information Systems, pages 8-17, 1993.

[14] D.B. Skillicorn, J.M.D. Hill, and W.F. McColl. Questions and answers about BSP. Technical Report PRG-TR-15-96, Computing Laboratory, Oxford University, 1996.Also in Journal of Scientific Programming, V.6 N.3, 1997.

[15] A. Tomasic and H. Garcia-Molina. Performance of inverted indices in sharednothing distributed text document information retrieval systems. In Second International Conference on Parallel and Distributed Information Systems, pages 8-17,1993.

[16] B. Ribeiro-Neto , Edleno S. Moura, Marden S. Neubert, and Nivio Ziviani Efficient Distributed Algorithms to Build Inverted Files , 1999 ACM.

[17] M. Marin. Index structures for distributed text databases