한국산업정보학회논문지 제7권 제5호 (2002. 12)

Memory Performance of Electronic Dictionary-Based Commercial Workload+

이 창 식*, 김 희 철*, 이 용 두*,
(Changsik Lee*, Hiecheol Kim*, Yongdoo Lee*)

요 약 인터넷의 급속한 성장에 따라 전자사전에 대한 트랜잭션 처리를 기반으로 하는 상용 응용 소프트웨어의 사용이 증가하고 있다. 그 전형적인 예로서 인터넷 검색엔진을 들을 수 있다. 본 논문에서는 고성능 전자사전의 구축을 위한 새로운 접근방법을 제시한다. 전자사전의 메모리 구현에 있어 트라이 데이터 구조를 사용하는 기존의 방식과는 달리, 본 논문에서 제시하는 방식은 다차원 이진트리 구조를 사용한다. 본 논문에서는 다차원 이진트리 기반의 전자사전이 ED-MBT(Electronic Dictionary based on Multidimensional Binary Tree)의 구현 내용과 실용적인 응용 소프트웨어에서 ED-MBT가 갖는 성능향상에 관한 세부적인 분석 결과를 제시한다.

Abstract long with the rapid spread of the Internet, a new class of commercial applications which process transactions with respect to electronic dictionaries become popular. Typical examples are Internet search engines. In this paper, we present a new approach to achieving high performance electronic dictionaries. Different from the conventional approaches which use Trie data structures for the implementation of electronic dictionaries, our approach uses multi-dimensional binary trees. In this paper, we present the implementation of our electronic dictionary ED-MBT(Electronic Dictionary based on Multidimensional Binary Tree). Exhaustive performance study is also presented to assess the performance impact of ED-MBT on the real world applications.

1. Introduction

WWW (World Wide Web) over the Internet opens a new era by making huge amount of information spreading worldwide be available to global Internet population. Management and retrieval of the information bring a new class of commercial application software. Among them are various search engines. These applications usually work as transaction processing with respect to huge irregular data contained HTML documents.

To support efficient transactions over these irregular data, documents are indexed by keywords. For normal Internet search engines, the number of keywords is usually order of tens of millions due to the huge number of HTML documents over the Internet. It is thus a crucial issue to implement a runtime electronic dictionary which is efficient for keyword searches. In relation with the huge number of keywords, the efficiency in terms of memory occupation of the runtime electronic dictionary is an additional issue.

The contents of HTML documents over the Internet changes very frequently. Recent investigation suggests that over 10% of HTML documents are either discarded or modified per weeks. Moreover new documents keep upload over the Internet. These frequent changes enforce that the contents of electronic dictionaries should also be kept updated. Indeed, the Internet search engines update periodically its database anew by gathering HTML documents over the Internet. This update procedures are mostly made manually by system administrators. Somehow, it is necessary to make the procedure be done

+ This work was supported by 1999 Research Fund from Daegu University.

* Professor, School of Computer and Communication Engineering, Taegu University.

automatically. This automatic procedure, which we will call dynamic indexing, modifies the runtime electronic dictionary and its indexed documents dynamically while search engines provide their services. To support dynamic indexing, it is essential to make efficient update, e.g., addition and deletion of keywords, of the runtime electronic dictionary. Therefore, it is another crucial issue for electronic dictionaries to support highly efficient dynamic update.

Even though the implementation of electronic dictionaries over the Internet applications is challenging with raising performance, memory efficiency and flexibility issues that we point out earlier, few studies are made. Most current applications just use schemes designed for document management systems which require

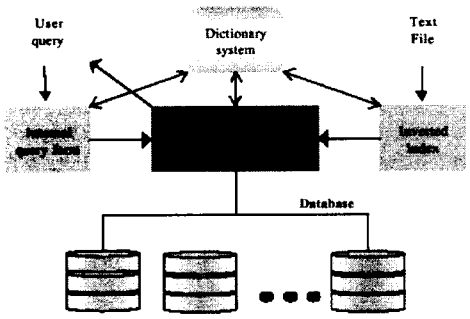


Figure 1. The organization of a Internet search engine

usually small or medium scale electronic dictionaries.

In our research, we explore performance, memory efficiency, and flexibility issues of electronic dictionaries for the Internet applications. To address the issues, we developed a new scheme to implement electronic dictionaries. It uses multidimensional binary tree to represent the runtime electronic dictionaries. The electronic dictionaries thus designed is called ED-MBT (Electronic Dictionary based on Multi-dimensional Binary Tree). This paper presents the theoretic aspect and performance of ED-MBT implemented in a commercial search engine.

The rest of the paper is organized as follows. In section 1, existing schemes for electronic dictionaries are

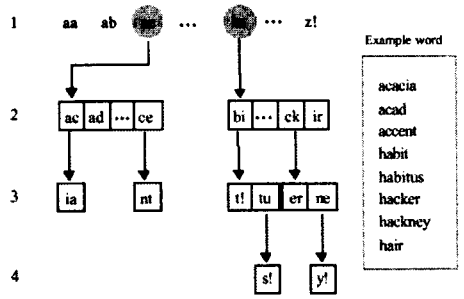


Figure 2. An example Trie-based electronic dictionary

presented as background. In section 2, ED-MBT is presented. In section 3, the result of the performance evaluation is presented. Finally, in section 4, the concluding remarks and future research issues are offered.

2. Background

Transaction processing of Internet applications are made over various forms of databases. Search engines, for example, collect HTML documents over the Internet and extract keywords from them. They then construct a database which contains the contents of all the documents indexed by keywords. Based on the database, they provide search services for the general Internet population[7]. The system components of Internet search engines are depicted in Figure 1

In services of Internet search engines, a user query is usually made by keywords. Given a keyword, the system searches the electronic dictionary for the keyword. Provided the keyword is a legal one, some information is retrieved from the electronic dictionary for the keyword. Using the information, the system locates the documents containing the keyword and returns them to the user.

This procedure indicates that transactions triggered by user queries are made with respect to the electronic dictionary. As pointed out earlier, the number of keywords used in the Internet search engine is tens of millions because nouns such as people's name must be accepted as keywords. Therefore, electronic dictionaries in such applications have critical impact on the performance of the Internet search engine.

Previously several schemes are used to build electronic dictionaries. The first scheme uses B+ trees to represent dictionaries. In this scheme, a node represents each keyword. The key value used for a node is calculated by using the characters of its keyword. This scheme is conceptually very simple and provides highly flexible deletion and addition of keywords. Moreover, storing the dictionaries in file, which is normally required for a system cold start is also efficient due to its inherent feature of B+ trees[3,5].

In this scheme the time complexity of each search operations of O(log n), where n is the number of keywords. For the internet search engine, as n is order of ten millions, the value of O(log n) becomes very high compared with other schemes which will be discussed shortly. However, the critical problem is that this scheme suffers intolerably poor memory efficiency because the number of memory words allocated per key must be the same one required for the largest keyword in terms of character counts. This drawback makes the scheme practically obsolete. In a search engine, a user query contains normally a string of characters and thus it is required to extract the keyword of maximal size from the string by some syntactic analysis. Normally, this analysis contains more than one dictionary searches for some substrings. With B+ tree schemes, these searches are very inefficient because each search is to made independently with respect to the whole B+tree[9,10].

The second scheme uses hash tables to represent dictionaries. In this scheme, a hash value calculated for a given keyword by using a hash function determines an entry in the hash table. This scheme is very efficient in terms of memory usages because only the hash table is required to be resident in memory. It also provides high performance in locating a keyword to its corresponding entry due to the inherent feature of the hash data struction[4].

However, it has critical problems in terms of performance and flexibility. First, as keyword has of variable character length with the combination of characters of a given character set (e.g., ASCII alphanumeric code) high frequency of hash collisions is indispensable in electronic dictionaries. Because it is entailed to make linear search over the list of colliding elements, collision processing is very costly even though the colliding elements are resident in memory. Second, is not flexible to make addition or deletion of a keyword due to the side effect related with making a permanent copy of colliding entries in files. That is, because the number of colliding entries are of variable length, the file management to store those entries is extremely inefficient. In the worst case, the whole file must be rebuild[10].

The third scheme uses the concepts of trie data structures (Figure 2). In this scheme, a level in the tree is associated with the character position in keywords[1,2,6]. For example, the root node is associates with the first character position of the keywords. Each node contains <[char, link]+> where char is the field for a character and link is a field for a child node. The characters in the child node are candidate characters that can follow the partial word formed along the path from root to the node containing char. A number of variation are found for this trie-based implementation of electronic dictionaries.

This scheme is featured with an incremental character by character search from the first character. This incremental search is beneficial during the resolution of keyword generation which extracts a maximally matching keyword from a character string; in the previous two schemes, we need to searches repetitively for each substring.

Overall, this scheme is very efficient. However, this scheme retains a problem which degrades the performance due to the requirement of the linear search over possible candidates inside a node. On the implementation side, because the amount of memory words required for each node is different, it requires a very complex data structure[8,10]. Furthermore, it has a critical problem in the addition and deletion of a character associated with each node. Therefore, dynamic update is not virtually possible without experiencing intolerable overhead.

3. Electronic dictionary based on multi-dimensional binary trees.

The survey of previous schemes allow us to bring the following requirement for the electronic dictionaries. First, the performance of a keyword search must be high for Internet applications such as search engines. Second, the dictionary must provide character-wise incremental search to efficiently support keyword extraction from a character string because keyword extraction is an essential step in the front of query processing. Third, the memory requirement must be small. Finally, the update of keywords for both the runtime data structure and disk files must be supported without incurring high overhead. Assessed by each requirement, none of the previous three schemes satisfy all the four requirements.

Table 1. Example keyword for ED-MBT

halfmast, halfpenny, halftone, hammer, hand,handrail, handwok, handwriting, harbor, harvard, harvesting, harvestmoon, harvest, hawk

Subject to the above four requirements, we thus developed a new scheme by taking multidimensional tree as the representation of electronic dictionaries. The electronic dictionary thus obtained is called ED-MBT (Electronic dictionary based on multidimensional binary trees).

ED-MBT is represented by a tree in which each node has a key value and three child nodes. Formally, let its content be <key, left, center, right>, where key is the key value, left is a field for the left child node, center is a field for the center node, and right is a field for the right child node.

In ED-MBT, the parent node, left child and right child nodes have the same semantics as in normal binary trees. On the other hand, the newly introduced center node has semantics associated with characters following in the next character position of a keyword. Removing edges connecting a parent node to its center node, a set of disjoint binary trees remain. It is defined that each binary tree is a data structure for the set of characters which are in the same character position in keywords.

Given a parent node which is associated with kth character position of keywords, the subtree whose root node is the center node represents the substrings which can follow in keywords the character of the parent node. In the subtree, the binary tree formed by removing all the subtrees of each center node is thus a data structure for the set of characters which can appear just after the character of the parent node, i.e., in the (k+1)th character position of keywords.

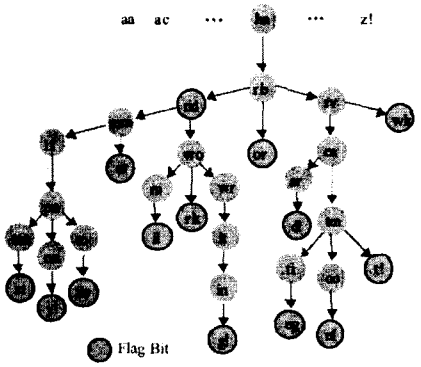


Figure 3. An example ED-MBT

In electronic dictionaries, each character in a given code system can appear as the first character of keywords. In a normal MBT, the binary tree for this first position must be a binary tree whose root is the root of the MBT. But, in ED-MBT, as the location of a character can usually be indexed by its code when the characters are stored in an array, the entire characters in the code is represented by a linear array, which will be called the root array. This help avoid binary searches for the first character in keywords.

Figure 3 shows an ED-MBT for the set of keywords in Table 1. In the ED-MBT, two ASCII characters are assigned to a key. The first key of the keywords is ha and is represented as an element in the root array. The second level keys, {lf, mm, nd, rb, rv, wk}, is represented by a binary tree. For each node n in the binary tree, the next level keys are formed in another binary tree whose root is the center child of node n. For example, for node m with key nd, each element of the set of keys {ra, wo, wr} is defined as a candidate for the next level key and the set is represented as a binary tree whose root is connected as a center child of node m.

Normally, each leaf node contains the last key of a

keyword. For example, node with key st contains the last key for keyword halfmast. In ED-MBT, if a keyword k1 is a prefix of another keyword k2, the node for the last

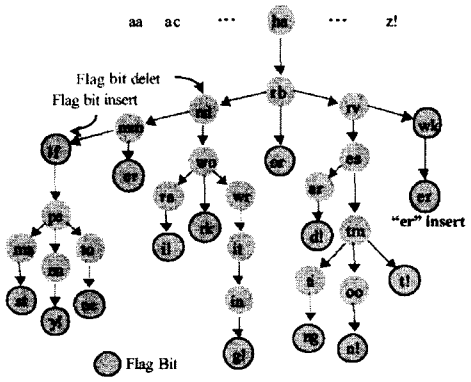


Figure 4. Insertion and deletion in ED-MBT

key of k1 has a center child to represent the remaining character string of k2, and is thus not a leaf node. For example, non-leaf node with key nd is the last node for keyword hand which is a prefix of keyword handworker. Therefore, in ED-MBT the non-leaf node for key nd has an information to indicate the resolution of keyword hand. The information is indicated by solid cirle in Figure 3.

Search a keyword in ED-MBT is incremental. For a first level key, the code value is used an index to the root array. Using the pointer contained in the indexed element of the root array, the root of the binary tree is identified. For the second level key, the binary tree is searched using normal binary tree search algorithms. Once a node is found for the second key, we take the center child node as the binary tree for the third level key. By repeating the procedure, we can search a keyword or resolve that the input keyword is not a legal keyword registered in the dictionary. Insertion and deletion of a keyword is straightforward because each level is a complete binary tree. Instead of a formal algorithm, we provide an example in Figure 3. The example shows insertion of hawker into the ED-MBT in Figure 4. Alto it shows insertion of half. The detailed description is not exposed in this paper due to space limitation.

Assessed from the requirements of electronic dictionary discussed earlier, ED-MBT satisfies all the requirements. First, ED-MBT offers a great opportunity for high performance in keyword searches because of its binary search without any linear searches. Second, ED-MBT provides character-wise incremental search to efficiently support keyword extraction. Third, the memory requirement is small because multidimensional trees are efficiently implemented by arrays of structures without fragmentations. Finally, benefiting from the characteristics of binary trees, ED-MBT efficiently support the update of keywords for both the runtime data structure and disk files.

4. Performance evaluation

To evaluate the performance of searches, we conduct experiments. First, given a set of test dictionaries, we measure the amount of memory required to build electronic dictionaries respectively for ED-MBT and ED-Trie (Electronic Dictionary based on Trie data structures). Subsequently, given a set of test input keywords, we measure respectively the total search time with respective to ED-MBT and ED-Trie dictionaries. Second, to analyze the characteristics of memory behavior of ED-MBT and ED-Trie on the state of the art computer systems, we measure the memory performance on a simulated machine.

4.1. Evaluation methodology

The performance evaluation is made with focus on the memory requirement and the performance of search operations. The performance of ED-MBT is compared with ED-Trie. In the implementations, ED-MBT and ED-Trie are used as shown respectively Figure 3 and Figure 2.

To make the evaluation realistic, we use the dictionary used in the Kachi Internet search engine which provides a commercial service for Korean Internet population. Using 260 million keywords of the Kachi dictionary, we made 9 kinds of test dictionaries whose keyword size are

respectively 60K, 300K, 600K, 900K, 1200K, 1500K, 1800K, 2100K, and 2400K. The 60K dictionary contains nouns in Korean dictionaries. For the remaining test dictionaries, keywords are selected evenly from entire 260 million keywords, not extracting some part. The average characters per keyword is about 4.80. Table 2 shows the keyword size, file size and average character size per keyword respectively for each of the above nine test dictionaries.

Table 2. Comparison of memory requirement

	Word count	Dic size (Mbyte)	Average syllable	MBT size (Mbyte)	Trie Size (Mbyte)	
Dic_1	62,128	0.40	3.42			
Dic_2	364,696	3.17	4.56			
Dic_3	662,074	6.10	4.83			
Dic_4	962,070	9.03	4.92			
Dic_5	1,262,069	12.00	4.98			
Dic_6	1,562,066	15.16	5.08			
Dic_7	1,862,062	18.23	5.13			
Dic_8	2,162,060	21.37				
Dic_9	2,462,059	24.52				

Table 3. MBT & Trie search time

Dic Name	ED - MBT Search Time			ED - Trie Search Time			Trie/ MBT
	User	System	Total	User	System	Total	
Dic_1	23.79	0.66	24.45	53.53	0.69	54.22	
Dic_2	28.53	0.55	29.08	92.50	1.06	93.56	
Dic_3	32.66	0.52	33.18	119.65	1.14	120.79	
Dic_4	33.74	0.74	34.48	141.33	1.37	142.70	
Dic_5	34.91	0.77	35.68	154.09	1.69	155.78	
Dic_6	35.67	0.78	36.45	167.96	1.34	169.30	
Dic_7	36.63	0.94	37.55	174.37	1.36	175.73	
Dic_8	37.24	0.86	38.10	186.68	1.36	188.04	
Dic_9	37.18	0.83	38.01	195.68	1.38	197.06	
					Average		4.10

In addition, to obtain keywords used as the input for the keyword searches over test dictionaries, we generate traces of queries from the commercial services of the Kachi Internet search engine. The number of input keywords thus obtained is 8,410,000. The average number of characters per input keyword is 3.43. The input keywords are used in evaluation the search time.

4.2. Memory usage

Based on the C programs implementing ED-MBT and ED-Trie, we collect the total runtime memory size required to represent for each of 9 dictionaries. Table 2 shows the result. For each dictionary, it shows the number of keys and the memory size to implement the dictionary. On average, bytes per key is about 26 for ED-Trie and 16 for ED-MBT. This indicates that in the representation of a key ED-MBT is more efficient by 62% ((26-16)/16*100) with respect to ED-Trie. The total amount of memory required to represent a dictionary respectively for ED-MBT and ED-Trie is shown in Table 2. The table shows that on average ED-MBT is more efficient in its memory requirement by about 56 percent than ED-Trie.

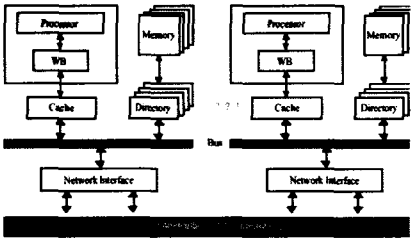


Figure 5. RSIM multprocessor simulation model

4.3. Search performance

To evaluate the search performance, we use the set of input keywords whose keyword size is 8.4 million. The evaluation is made by measuring the execution time to search the entire input keywords sequentially respectively for the C implementations of ED-MBT and ED-Trie on Sun UltraSparc workstation. On each implementation, the execution time is measured for each of 9 dictionaries. The total execution for searching 8.4 million test keywords is measured for the system time and the user time (Table 3).

Overall, ED-MBT is very faster than ED-Trie per keyword search. Table 3 shows a graph which shows the performance between ED-MBT with ED-Trie in terms of search time. For Dict-1, ED-MBT is 2.22 times faster than ED-Trie and for Dict-9 ED-MBT is 5.18 times faster. For all nine dictionaries, on everary the speedup of keywad searches in ED-MBT over in ED-Trie is 4.10.

As we use the same input keywords for all the 9 different dictionaries, the result in Table 4 reveals the scalability of each scheme with respect to the dictionary

size. According to the table, in ED-MBT the search time per keyword is not affected much by the size of the dictionary. On the other hand, in ED-Trie the search time per keyword increases almost linearly as the size of the dictionary becomes larger. Indeed, the ratio of Dict-9 to Dict-1 in term of search time (Dict-9/Dict-1) is 3.65 in ED-MBT, while it is 1.6 in ED-Trie. This indicates that ED-MBT is much suited for large-scale dictionaries due to its higher scalability over dictionary size.

Overall, the table shows that ED-MBT is superior to ED-Trie both for scalability over the size of the dictionary as well as search performance.

Table 4. Cache hit-rate and execution cycles

Associate	Line size	ED-MBT		ED-Trie	
		hit rate (%)	Cycles	hit rate (%)	Cycles
2 - way	16	94.89	4.16e+06	93.41	1.13e+07
	32	95.67	4.11e+06	91.78	1.21e+07
	64	98.08	4.10e+06	95.22	1.16e+07
	128	96.50	4.11e+06	97.62	1.16e+07
4 - way	16	95.19	4.12e+06	93.99	1.12e+07
	32	95.93	4.08e+06	92.35	1.20e+07
	64	98.29	4.06e+06	95.54	1.15e+07
	128	96.80	4.05e+06	97.91	1.15e+07
8 - way	16	95.31	4.11e+06	94.08	1.11e+07
	32	96.00	4.07e+06	92.47	1.19e+07
	64	96.37	4.05e+06	95.61	1.15e+07
	128	96.80	4.02e+06	98.01	1.14e+07

4.4. Memory performance

Modern computer systems provide a number of hardware supports to get optimizations which exploit the execution behaviors of applications. One of important classes of them is to minimize memory latency. The hardware supports are based on the analysis of memory behaviors such as locality and access patterns of memory operations. These analyses are mostly focused on scientific applications and thus their memory behaviors are relatively well understood. On the other hand, only a few studies are made for analyzing memory behaviors of commercial workload such as transaction processing with respect to databases[12,13]. Furthermore, for directory-based transaction processing no previous studies are found to analyze the memory behaviors. It is quite sure that algorithmic excellence of ED-MBT contributes to higher performance of ED-MBT than ED-Trie. But we still need to understand the memory behavior of ED-MBT to benefit from the architectural optimization such as cache memories.

In this experiment, we explore the memory performance of electronic dictionaries. The study is based on the software simulating of keyword searches respectively for ED-MBT and ED-Trie. Through simulating, we gather the cache performance respectively for various configurations. The cache performance is compared between ED-MBT and ED-Trie.

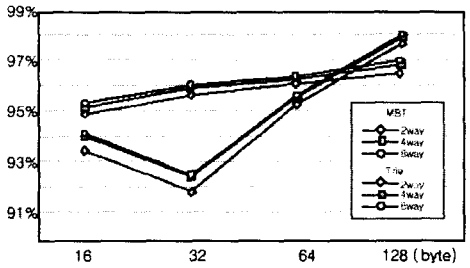


Figure 6. Cache hit ratio

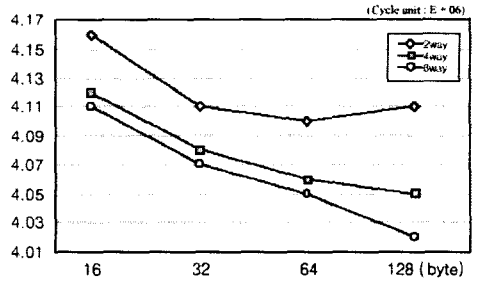


Figure 7. ED-MBT execution cycles

The software simulator that we use is RSIM (Rice Simulator for ILP Multiprocessors)[11]. In our experiment, we use the single processor configuration of RSIM, while this simulator is developed to simulate multiprocessors. RSIM provides a means for very accurate measurements of modern processors because its processor model is based on modern MIPS R10000 processor architectures (Figure 5).

In this experiment, we use the input set which consists of 10,000 keywords. The size is far smaller than the previous experiment to evaluate the search time. The reason for using smaller input keywords is that for large

input keywords the simulation takes too long to manage the simulation work. However, we believe that the result obtained for 10,000 keywords still provide sufficient information for the analysis of the cache performance of ED-MBT and ED-Trie. Indeed, through some experiments we see that larger input sizes than 10,000 keywords shows only negligible difference in the characteristics of the cache behavior both in ED-MBT and ED-Trie.

Given 256K bytes of cache, the evaluation is made with respect to two cache parameters: cache associativity, and cache line size. In addition, the number of cycles to execute 10,000 keyword searches is measured. The result is shown in Table 4.

Figure 6 shows the cache performance. In terms of the cache associativity, both ED-MBT and ED-Trie show some expected characteristics in their hit ratios. That is, as the associativity becomes larger, the hit ratio increases for both schemes. As the difference between cache hit ratios are below 1 percent, the cache associativity does not make any noticeable impact on the cache performance. In terms of the cache line size, ED-MBT shows an expected results. That is, as the cache line size increase, cache hit value increases. On the other hand, in ED-Trie the cache hit ratio drops when the line size is 32 bytes. In ED-Trie, the memory words required to implement a key is 26 bytes. The difference between the node size of 26 bytes and the line size of 32 bytes causes more fragmentation than in the other configurations of the line size. In terms of cache hit ratios, ED-MBT shows higher value than ED-Trie except when the line size 128. A node in ED-Trie contains usually more than one key and an array of structure is used for the node. Given a set of candidate keys, when they are implemented in a array and the cache line size becomes larger, the locality tends to be higher than when they are implemented by binary trees. This is, therefore, the reason for lower hit ratio of ED-MBT when the line size is 128 bytes than that of ED-Trie. However, as a larger cache line causes larger block transfer time, the higher hit rate of ED-Trie for larger cache lines do not directly reflected in the execution performance. This point will be explained shortly.

To assess the cache performance from the view point of the cache line size, ED-MBT shows locality which are insensitive to the cache line sizes. This property can allow ED-MBT to benefit more from a wide range of configurations in the cache line than ED-Trie.

Figure 7 and 8 shows the execution cycles for 10,000 keyword searches respectively for ED-MBT and ED-Trie. Comparing with ED-MBT and ED-Trie, we see that the execution times obtained from simulation conform to the execution time obtained from direct execution in section 4.3. In ED-MBT, the execution cycles mostly decrease as the bock size decrease except when the cache uses two-way set associative mapping with the line size of 128 bytes. This indicates that up to 128 bytes block size the higher cache hit ratio in ED-MBT directly contributes to higher execution performance. On the other hand, according to the graph in Figure 8. the high cache hit ratio of ED-Trie obtained when the line size is 128 bytes does not contribute to the execution performance due to the large cost of block transfers between the cache and the main memory.

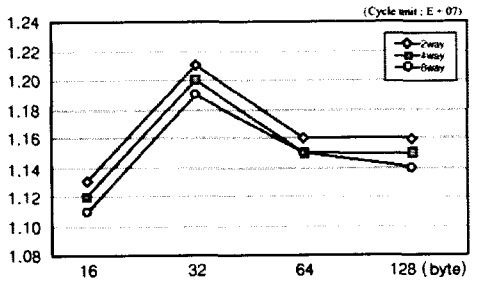


Figure 8. ED-Trie execution cycles

5. Conclusions

This paper presents a new scheme, ED-MBT, to build electronic dictionary based on multidimensional binary trees. Compared with the previous schemes, our ED-MBT is excellent by providing incremental searches and highly efficient dynamic update of runtime data structures. The performance of ED-MBT is very scalable with respect to the size of dictionaries which are usually of huge size for dictionary-based Internet applications such as Internet

search engines. In terms of raw performance, the average search time per keyword of ED-MBT is very smaller than Trie-based directories by 2.2 times for 60k dictionaries and by 5.18 times for 240k dictionaries. The performance analysis shows that ED-MBT retains both (1) algorithmic enhancement and (2) higher locality of memory access patterns over previous schemes, particularly Trie-based approaches.

Reference

[1] J . I. Aoe. "An Efficient Digital search algorithm by using a Double - Array Structure", IEEE Transactions on S/W Eng. Vol. 15. No. 9, pp. 1066-1077. 1989.

[2] J. I. Aoe, K. Morimoto, "An Efficient Implementation of Trie Structures", S/W Proc. and Exp. Vol. 22. No. 9. pp. 695-721. 1992.

[3] D. Corner, "The Ubiquitous B-Tree", ACM Computing Surveys, Vol. 11, No. 2, pp. 121-137, 1979.

[4] R. J. Enbody, H. C. Du, "Dynamic Hashing Schemes", ACM Computing Surveys, Vol.20, No.2, pp. 85-113, 1988.

[5] William B. Frakes, Richard Baeza-Yates, "Information Retrieval Data Struct & Algorithms", PTR Prentice-Hall, 1992.

[6] Edward Fredkin, Bolt Beranek and Newman, "Trie Memory", Communications of the ACM, Vol.3, pp. 490-499, 1960.

[7] Jong-Back Kang, Sung-Hun Kim, Won-Kyo Jeong, Yong-Doo Lee, "An Indexing Algorithm for Korean Words using DFAs", Proceedings of The 6nd Korea Information Processing Society, Vol.3, No.2, pp1014-1019, 1996.

[8] Cheol-Su Kim, Woo-jeong Bae, Yong-seok Lee, Jun-ichi Aoe, "Construction of Electronic Dictionary Using Double-array Trie Structure", Journal of Korea Information Science Society, Vol.23, No.1, pp.85-94, January, 1996.

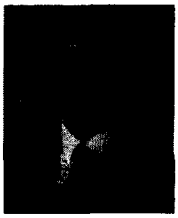
[9] D. Knuth, "The Art of Programming", Vol. 3, Sorting and Searching, Addison-Wesley Publishing Co. Inc. 1973.

[10] Seung-Sun Lee, Ju-Won Song, Wan-Sup Cho, Kyu-Young Whang, Ki-Sun Choi, "A Database Index Structure for the Korean Electronic Dictionary", Journal of Korea Information Science Society, Vol.22, No.1, pp.3-12, January, 1995.

[11] Vijay S. Pai, Parthasarathy Ranganathan, Sarita V. Adve, "RSIM: An Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors", In Proceeding of the 3rd Workshop on Computer Architecture Education, February, 1997.

[12] Trancoso, P., Larriba-Pey, J., Zhang, Z. and Torrelas, J., The Memory Performance of DSS Commercial Workloads in Shared-Memory Multiprocessors, Proc. of 23rd Ann. Int. Symposium on Computer Architecture, 1996.

[13] Thakkar, Shreekant and Sweiger, M., Performance of an OLTP Application on Symmetry Multiprocessor System, Proc. of the 1990 Int. Computer Architecture Symposium.

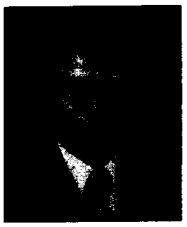


이 창 식 (Chang-sik Lee)

1975 : kyungpook University (Electronic Eng. B.S.)
1979 : kyungpook University (Computer Eng. M.S.)
1999 : Konkuk University (Electronic Eng. Ph.D.)
1981 ~ : Daegu University, Professor
Interest: Microwave, Digital communication

김 희 철 (Hicheol Kim)

1983 : Yonsei University (Electronic Eng. B.S.)
1991 : University of Southern California (Computer Eng. M.S.)
1996 : University of Southern California (Computer Eng. Ph.D.)
1997 ~ : Daegu University, Assistant Professor
Interest : GRID, Computer Architecture, Parallel Processing, Compiler



이 용 두 (Yongdoo Lee)

1975 : Hankuk Aviation University (Communication Eng. B.S.)
1982 : Yeungnam University (Computer Eng. M.S.)
1995 : Hankuk Aviation University (Computer Eng. Ph.D.)
1982 ~ : Daegu University, Professor
Interest : GRID, Computer Architecture