The Telephone Directory Enquiry System of Hong Kong

K.P. Chow, T.W. Lam, and K.H. Lee
Department of Computer Science
University of Hong Kong, Hong Kong
Email: {chow, twlam, khlee}@cs.hku.hk

Abstract

This paper is concerned with the design and performance of the telephone directory enquiry system newly adopted in Hong Kong. This system maintains three million telephone records and supports over forty thousand enquiries per hour at the peak. In the Hong Kong society the uses of English and Chinese (in particular, Cantonese) has been blending in a thrust of exciting language culture, giving rise to a variety of telephone enquires that traditional B-tree or hashing based telephone directory enquiry systems fail to handle. The efficiency and flexibility achieved by the new system stem from hosting all indexing data structures in the main memory; these data structures occupy about half giga-bytes and would have been considered too expensive to be placed in the main memory in the past.

1 Introduction

In Hong Kong, there are about three million telephone records and online telephone directory enquiry always has a big demand, especially in the business sector. At present, the Hong Kong Telecom Company Limited provides a 24-hour online enquiry service. Everybody can phone to the enquiry center to ask the operator to search the telephone records through the computer. At the peak, the enquiry center receives over forty thousand enquiries per hour.

The previous telephone directory enquiry system used in Hong Kong was developed fifteen years ago. It was based on an implementation of B-trees, in which telephone records are indexed by the names of customers in English (or Chinese). Such an enquiry system is similar to other systems used in countries with English as the primary language and it can serve swiftly for enquires that specify clearly the whole name of customers or the leading characters. However, due to the intermix of Chinese and English language culture in Hong Kong, the number of enquiries that fail to provide the whole name of customers or the leading characters is much more than expected. As to be explained in Section 2, there are indeed many cases in which the operators have to use a trial-and-error basis to search the telephone records, generating a number of queries to the computer; and in the worst case the required records could not be found. One of the goals of developing a new system is to support very flexible queries so that the operator can handle an enquiry by invoking a single query to the computer.

Apart from supporting flexible queries, we must also ensure the shortest possible response time to each individual query. Existing database systems such as Oracle and Sybase cannot satisfy such requirement. They are designed for managing large data sets and can serve straightforward queries efficiently. But for those more complicated queries such as searching for incomplete names, the response time is not acceptable. In the new telephone directory enquiry system, the indexing data structures of the telephone records are indeed placed in the main memory. This requires about half giga-bytes of memory. Nowadays machines with even one giga-bytes of main memory are commercially available at a reasonable cost. Putting all indexing data structures into the main memory speeds up the searching algorithms drastically and makes it feasible to support those complicated queries. More importantly, from the viewpoint of system design, the whole system becomes simpler to implement and manage. We believe that in the near future more database systems that are traditionally built on secondary storage would be redesigned to run on the main memory; Some complicated data structures and algorithms designed to cope with secondary storage will also be simplified.

0-8186-7638-8/96 $05.00 © 1996 IEEE

The remainder of this paper is organized as follows: Section 2 describes the characteristic of telephone directory enquires in Hong Kong and lays down the requirements of the queries we expect the new enquiry system to handle. Section 3 studies the architecture of the new enquiry system. Section 4 shows the details of the indexing data structures of telephone records, that are to be kept in the main memory. These data structures adopt a uniform treatment to both Chinese and English words and admit a simple and efficient algorithm to search the telephone records. Section 5 highlights the way the indexing data structures is updated while a number of enquiries are served at the same time. In the last section, the performance of the new enquiry system is discussed.

2 Characteristics of Telephone Directory Enquires in Hong Kong

Most of the telephone directory enquiries are actually related to the business. When a business is set up in Hong Kong and applies for a telephone line, it is only required to supply a name of the business in English and the Chinese name is optional. This is probably due to the historical development of Hong Kong. Moreover, there are many international businesses in Hong Kong which do not have official Chinese names. Apparently, there should not be any problem if every enquiry can specify the name of the business in English. Yet about 98% of the residents in Hong Kong are Chinese speaking a Chinese dialect called Cantonese. Though English is an official language and a compulsory language subject at school, most residents cannot speak fluent English. When the operator receives a call, it is very often that the enquirer fails to provide the full name of the business in English, but can probably remember one or two keywords of the business in English, and the rest in Chinese. Thus, an enquiry is often composed of only Chinese words or a mix of English and Chinese words.

Of course, one may expect the operator to translate an enquiry to English. However, this creates a lot of problems because there is no unique way to do the translation and the Chinese and English names of some businesses may actually have no relationship. It is more desirable to build a telephone directory enquiry system that supports enquiries in Chinese, English, or a mix.

Another interesting point is that an enquirer, as well as the operator, often fails to pronounce some English words or spell the words correctly. Thus, the operator often has difficulty in entering the exact spelling of some English words. To remedy this situation, the operator usually puts down only the prefix or suffix of an English word to avoid mismatch due to wrong spelling. For example, many residents cannot pronounce and spell the word "Shangrila" correctly; it is not surprising to find the operator to enter a query in the form of "Shan- Hotel" or " -la Hotel".

In Hong Kong, a business can be set up in no time. There are many businesses with the same names. For instance, there are over a hundred businesses with the same name "Hung Fat". To narrow the search, the operator always asks for address information such as the names of the district, street, or building in English or Chinese.

In summary, a query prepared by the operator is comprised of up to four fields: English name, Chinese name, English address, and Chinese address. Each field itself contains a sequence of keywords. Note that an English keyword can appear in the form of a prefix and/or suffix, but a Chinese keyword is simply a single character. To process such query, the enquiry system must find the record(s) that matches all these fields. Traditional databases which organize the telephone records according to the full English names or the full Chinese names would obviously have a poor performance.

3 Architecture Overview

The new enquiry system is a client-server system consisting of six server machines (Unix machines) and about three hundred client machines (personal computers) connected through a network.

Client: A client machine is an ordinary PC. There is not much computation done on it. When a client machine starts up, it is assigned to be served by one of the six server machines. Whenever a query is entered through a client machine, a message is sent to the assigned server machine, which will process the query and return the result in a message. The client machine waits for this result message to arrive, then formats and displays the result accordingly. The client-server message passing mechanism is implemented by Remote Procedure Calls [3].

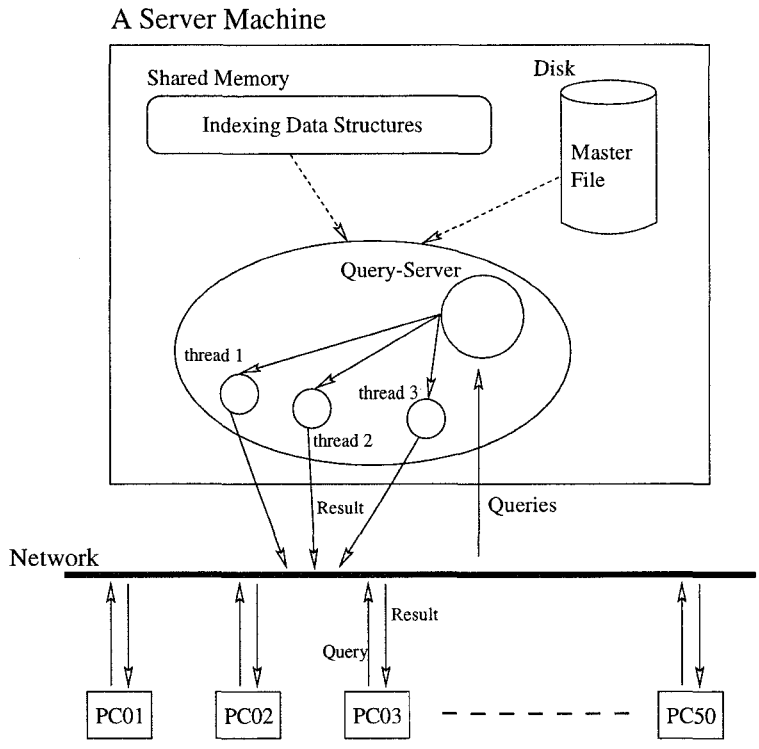

Figure 1: System architecture

Server: The six server machines are identical, running in parallel to even out the work load. Each server machine is a HP9000-K200 machine with 768 megabytes of main memory. In each server machine, there is a process called `Query-Server` which contains the program code of the searching algorithms described in Section 4. `Query-Server` is a multi-threaded process [5]. Whenever it receives a query from a client machine, it generates a thread (with the program code shared) to handle that request. This new thread executes the searching algorithms and sends the result in a message to the client machine. The main memory of a server machine is basically occupied by the indexing data structures of the telephone records and the threads generated by `Query-Server` (details of memory utilization will be discussed in Section 4). Figure 1 depicts the architecture of the new enquiry system.

Telephone records and indexing data structures: Each server machine keeps a copy of the master file of the telephone records in its hard disk. The master file is a plain file; each record is accessed by a unique record number (which is in the range between one and the total number of records). All indexing data structures to these records are kept in the main memory; they are built from scratch when the server machine starts. These data structures are shared and read simultaneously by the threads generated by `Query-Server`. From the viewpoint of a thread, a search in these shared data structures results in a list of record numbers that correspond to those records needed. The actual telephone records are retrieved from the master file eventually and sent to the client machine.

Daily Update and weekly reorganization: Everyday there are approximately 6,000 updates of the telephone records. Each update is either an insertion or a deletion of a record (a request of change is treated as a deletion followed by an insertion). The update requests are serialized at the enquiry service center and sent to all server machines through the network. Each server machine has one single process dedicated to serving such request. Upon receiving a request, this process updates the master file and indexing data structures in real time. A very important issue in designing the algorithm for updating of the indexing data structures is not to lock any memory location. This is to ensure that all enquiries can be processed concurrently and correctly. Details of the algorithm are discussed in Section 5. To simplify and

speedup the updating process, records that are supposed to be deleted are only "marked". After a while, many obsolete entries may be left in the hard disk and the main memory. Thus, a global reorganization of the master file and the indexing data structures is performed once a week. During the reorganization, no enquiry is supported by the server machine. All client machines connected to this server machine are switched to other server machines. To minimize the disturbance to the enquiry system, the reorganization is carried out at mid-night and at most one server machine is reorganized in one day.

4 Data structures and searching algorithms

To process a query described in Section 2 efficiently, we need to minimize the number of disk accesses to the master file of telephone records. This is achieved by building a number of indexing data structures (more precisely, tries) in the main memory to capture all name and address information of all telephone records. A query is processed by a search in these tries, which produces a list of record numbers referring to the required records. Only these required records are retrieved from the hard disk.

4.1 Tries—a review

Tries [4, 6, 7] are popular data structures for representing a set of words. Unlike AVL trees, B-trees, or other comparison-based data structures, tries can take advantage of the fact that the word can be decomposed into a sequence of characters, and thus support very efficient search for a word or a prefix.

A trie is simply a rooted tree in which every node (except the root) contains a character and a terminal flag. Figure 2(a) depicts a trie representing seven words. Note that the path from the root to a node defines uniquely a sequence of characters. For a node with its terminal flag turned on, it is called a terminal node and its path from the root corresponds to a distinct word represented by the trie. In other words, the set of words represented by the trie is defined by its terminal nodes.

Given a word w, it is straightforward to check whether w is one of the words represented by a trie. We simply follow a path in the trie labeled with w. If the trie does represent w among others, the path should eventually end at a terminal node. To ease our discussion, we denote MATCH(w) this terminal node. To find all the words that contain w as a prefix, we can examine the terminal nodes inside the subtree rooted at MATCH(w) using any tree traversal algorithm. We also denote PREFIX(w) the set of such terminal nodes.

4.2 Trie-based indexing data structures

The indexing data structures of the new enquiry system basically consists of four tries, corresponding to the English names, English addresses, Chinese names, and Chinese addresses of the telephone records. These tries are of the same structures. Our discussion focuses on the trie for English names. Denote this trie T_e. T_e represents every word that appears as part of the English name of some telephone records. Each terminal node in T_e is augmented to keep a list of record numbers. If the i-th telephone record contains a word w then the list for the terminal node representing w contains i. An example of such an augmented trie is shown in Figure 2(b). Note that we do not require the list of record numbers to be sorted in any order. The list of record numbers associated with each terminal node is implemented as a linked list of blocks, each containing up to ten record numbers. This list will be updated dynamically (see Section 5). We choose the block size as ten instead of one in order to save space for pointers.

When a server machine starts, T_e is built from scratch. We start off with an empty trie containing a dummy root node. We retrieve the telephone records from the master file one by one, and attempt to insert every word in the name of a telephone record into T_e. Details are as follows: Suppose we are examining the i-th record that contains a word w in its name. Step 1: We start from the root of T_e and trace the path labeled with w. If we fail to proceed before reaching the end of w, we create a chain of non-terminal trie nodes in T_e to store the missing characters of w. Step 2: Let α be the node in T_e of which the path from the root is labeled with w. If the terminal flag of α was off before, it is turned on now and α is associated with a list containing one record number, i.e. i. Otherwise, we simply append the number i to the existing list associated with α; if the last block of the list has been full, a new block is created.

Given a suffix "–x", searching T_e for all the words

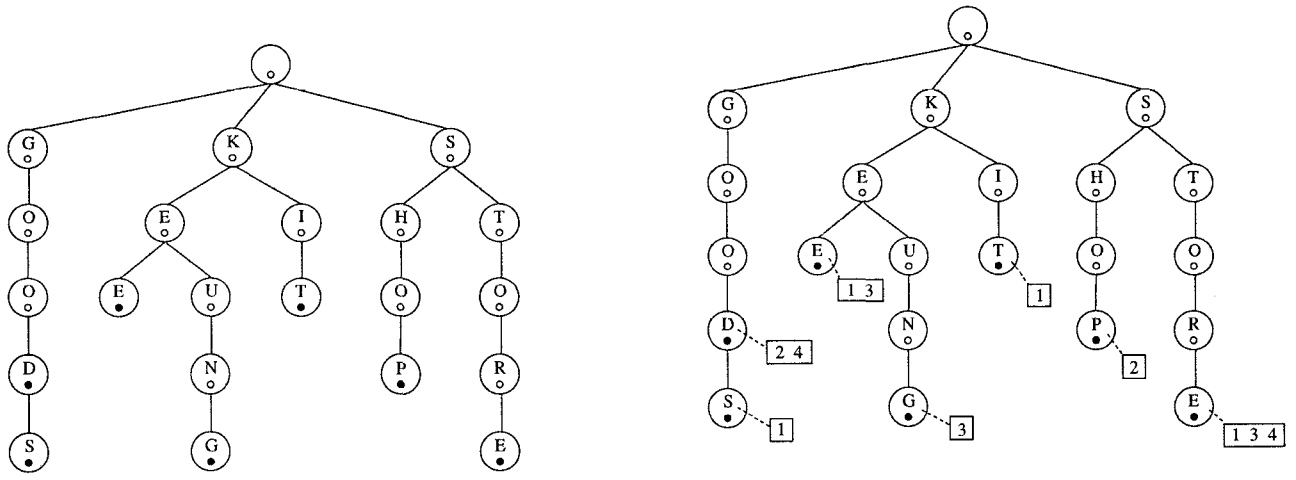

Figure 2: (a) A trie representing the words GOOD, GOODS, KEE, KEUNG, KIT, SHOP, STORE. The small circle in each node indicates the status of the node; a terminal node has this circle filled. (b) Each terminal node augmented with a list of record numbers.

that contain x as a suffix is very inefficient. To speed up the computation, we store an additional trie T'_e representing the reversed words of all the names. To handle a suffix "$-x$", we search T'_e instead of T_e for all the words that contain the reverse of x as a prefix. To reduce the space for storing T'_e, every terminal node of T'_e shares the list of record numbers with the corresponding terminal node of T_e. For example, the list associated with the terminal node representing the word "SHANGRILA" in T_e should be the same as the list associated with the terminal node representing the word "ALIRGNAHS" in T'_e; thus, T_e and T'_e can share the same list. In addition to T'_e, there is also a reversed trie for English addresses.

A Chinese word is represented by a two-byte code and can be treated as an English word consisting of two letters. The tries for the Chinese names and addresses are basically the same as the tries discussed above, but the depth of these tries is exactly two. Note that a Chinese keyword is a single character and we do not need a reversed trie for searching the suffix of a Chinese keyword.

4.3 Searching algorithms

As a warm-up, we first consider a simple query Q consisting of k keywords $w_1, w_2, \cdots, w_k$ of a name, each w_i is in the form of a complete word or a prefix.

For any $1 \leq i \leq k$, we search T_e for the keyword w_i. Depending on whether w_i is a complete word or a prefix, we identify the required terminal nodes in T_e, i.e. MATCH(w_i) or PREFIX(w_i). Let L_{w_i} denote the union of the lists of record numbers associated with these terminal nodes. Then, the answer to the query Q should be the record numbers in $L_{w_1} \cap L_{w_2} \cap \cdots \cap L_{w_k}$. It remains to show how to efficiently compute each L_{w_i}, which involves a number of list union operations, and find the intersection of all the L_{w_i}'s.

The problem of finding the intersection (or the union) of lists of arbitrary numbers is a classical problem. It is unlikely that one can obtain an algorithm with time complexity better than $O(n \log m)$, where m is the number of lists and n is the total number of numbers in the lists. In our case, the number of lists involved in the intersection problem is usually very small; however; for a particular w_i that is a prefix, there may be a large number of corresponding terminal nodes in T_e, and we need to union a large number of lists.

Below, we show a two-pass algorithm for the above list union-and-intersection problem. It is very efficient and actually reads every number in a list at most twice. The improvement stems from the observation that the numbers in the lists are not arbitrary, they are always bounded by the maximum number of records in the master file (i.e. approximately three million). Therefore, our algorithm can afford to allocate a working array A of this size in the main memory. Though this array is huge, our algorithm never examines every entry of this array in brute force, and it even does not need to initialize every entry of this

array. Details are as follows:

Pass I: First, we traverse the lists associated with the terminal nodes defined by w_1, and set $A[j]$ to zero whenever the record number j is encountered. Second, we traverse the lists associated with the terminal nodes defined by w_2, if a record number j is encountered and $A[j]$ is equal to zero, we set $A[j]$ to 1. Similarly, we process the lists for $w_3, \cdots, w_k$ one by one. In general, when we traverse the lists for w_i, if a record number j is encountered and $A[j]$ is equal to $i-2$, we set $A[j]$ to $i-1$. Figure 3 gives an example illustrating the updating of the array A.

Pass II: Note that if a number j is in $L_{w_1} \cap L_{w_2} \cap \cdots \cap L_{w_k}$, $A[j]$ should be equal to $k-1$, but the converse may not be true (since A is not assumed to have been initialized properly). If we examine every entry of A to report all j's such that $A[j] = k-1$, it is very time consuming and may produce wrong results. The way we find the intersection of all L_{w_i}'s is as follows: We traverse the lists associated with the terminal nodes defined by w_1 again. For each record number j encountered, if $A[j]$ is equal to $k-1$, we report the number j.

Next, we highlight the way we handle more complicated queries. Consider a query Q that involves some English name keywords in the form of a suffix "–y". The search for the suffix "–y" is conducted in the trie T'_e instead of T_e. This also produces a number of lists of record numbers. The answer to the query Q can still be obtained by performing the union-and-intersection operation on the lists defined by all the keywords. For a query involving keywords in other fields such as English address, we search the corresponding tries for the lists of record numbers. At the end, the union-and-intersection operation is performed on all the lists defined by the keywords.

Notice that for some enquires, the enquirer may specify that the ordering of the keywords in the name field must be observed. In such case, our searching algorithm still proceeds as before, but after all the records have been retrieved from the master file, it performs an additional filtering to ensure only those records that observe the ordering of the keywords are returned.

4.4 Memory utilization

A server machine is equipped with 768 mega-bytes of main memory. The indexing data structures for the three million telephone records occupy about 500 mega-bytes. The rest of the memory is allocated to the threads of `Query-Server`. The memory usage of such a thread is dominated by the temporary array A mentioned above. We assume the maximum number of telephone records is 3.6 millions, and restrict the number of keywords in a query to be at most sixteen. The latter is reasonable as the name kept in a telephone record seldom contains more than ten words and the enquirer provides at most five words in most cases. Therefore, each entry of A stores a value in the range from zero to fifteen and can be represented by a half-byte. The memory requirement for A is 3.6/2 = 1.8 mega-bytes of memory. Together with other dynamic variables, a thread of `Query-Server` needs about 2 mega-bytes of memory. On average, there are fifty client machines connected to a server machine. Even all the client machines submit queries at the same time, about 100 mega-bytes of memory are sufficient to host all the threads of `Query-Server`.

5 Online updating

As mentioned before, the new enquiry system supports online updating of telephone records. That means, the indexing data structures in the main memory are being updated while queries are processed simultaneously. The update requests are serialized and processed by the server machine one by one.

Below, we sketch the algorithms for updating of the indexing data structures. These algorithms do not require any locking of the memory, yet can always guarantee the correctness of a search.

Deletion: We maintain an array called DELETE-BITS in the shared memory. The size of this array depends on the maximum number of telephone records. The i-th entry of the array is equal to one if and only if the telephone record with record number i has been deleted. A request for deleting a record is processed simply by setting the corresponding entry in DELETE-BITS to one. The searching algorithms have to be modified with a filtering. That is, after performing the union-and-intersection operation to generate a list of record numbers, each record number is validated with the array DELETE-BITS.

Insertion: To insert a new record, we first append it to the master file, and assign the next available record number to it. Next, we should update the indexing data structures in the main memory. This

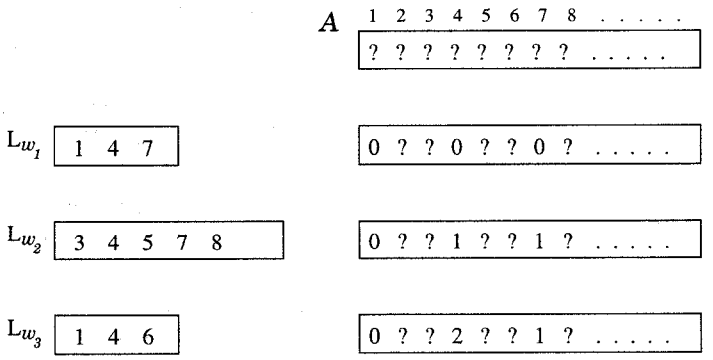

Figure 3: An example of Pass I

has to be done carefully as a number of threads of `Query-Server` may be searching these data structures at the same time.

Conceptually, what we need to do is to insert each word in the English name, Chinese name, English address, and Chinese address of the new record into the corresponding tries. The procedure should be the same as that of building the tries incrementally when the server machine starts. However, to guarantee a search to be performed smoothly at the same time, we make sure that while the tries are being updated, they never contain any dangling pointers and a traversal inside these tries will not be led to nowhere. The problem concerned can be abstracted as follows: There is a linked list on which several processes are traversing simultaneously, and to which a process wants to add a node. We do not care whether the processes will or will not encounter the new node during their individual traversals, but they must visit all other nodes on the list. The solution to this problem is very simple. We always ensure that the successor pointer in the new node has been set properly to a node in the list before we modify any successor pointer in the list to point to it. Based on the above idea, it is easy to write an insertion procedure for the tries that can ensure any search can proceed smoothly as before.

To avoid those threads of `Query-Server` that are generated prior to the insertion of a record producing inconsistent or even wrong result regarding this record, we require that when a thread is generated, it first memorizes the currently largest record number before executing the search procedure. During the search, any record number encountered greater than this memorized number is not reported.

6 Performance

To find out the response time of the server machine to a query, we have performed testing with a simulated environment under different loading conditions. We control the number of clients which send queries to the server machine repeatedly, as well as the inter-query time at each client, i.e. the amount of time between a client receiving the result of previous query and sending the next query. The queries used (totally, five thousands) are extracted randomly from the daily log of the "real" queries. Figure 4 shows the average response time (in seconds) of the enquiry system with respect to different combinations of number of the clients and inter-query time.

Note that an operator usually needs at least five seconds to communicate with an enquirer before submitting a query to the server, we are more concerned with the performance when the inter-query time is five or more. The above table shows that when there is no more than ten clients, the response time remains almost the same even the inter-query time is reduced to one second, this is because there is little chance of several queries being processed at the same time. When the number of clients exceeds twenty, the response time increases rapidly with more clients or shorter inter-query time. Nevertheless, under the load of fifty clients and five seconds of inter-query time, the average response time is only one tenth of a second. This is considered to be fast enough in a telephone directory enquiry system as the time for an operator to prepare a query is at least a few seconds.

In reality, after the release of the new enquiry system, the average time for an operator to serve an enquiry is estimated to have dropped by 30%. This achievement is not only due to the shortening of the

	number of clients						
	1	5	10	20	30	40	50
inter-query time = 10 seconds	0.038	0.043	0.044	0.047	0.051	0.059	0.074
inter-query time = 5 seconds	0.038	0.043	0.044	0.050	0.067	0.085	0.105
inter-query time = 1 seconds	0.038	0.049	0.076	0.152	0.324	0.671	1.112

Figure 4: The average response time (in seconds) of the enquiry system under different combinations of number of the clients and inter-query time.

response time to a query but also due to the availability of very flexible queries. With the new system, an operator seldom needs to invoke several queries to the server machine in order to serve one enquiry. Note that the time saved in invoking one less query per enquiry is much more significant than in improving the response time.

Acknowledgments: We are grateful to Hong Kong Telecom Company Limited, in particular, Cyril Leung, Hewlett-Packard Hong Kong Limited, and F. Chin and H.F. Hung of the University of Hong Kong.

References

[1] G.M. Adel'son-Vel'skii and E.M. Landis, An Algorithm for the Organization of Information, *Soviet Math. Doklady*, 3, 1962, pp. 1259-1262.

[2] R. Bayer and E.M. McCreight, Organization and Maintenance of Large Ordered Indices, *Acta Informatica*, 1, 1972, pp. 173-189.

[3] J. Bloomer, *Power programming with RPC*, O'Reilly & Associates, Inc., 1992

[4] E. Fredkin, Trie Memory, *Communications of the ACM*, 3, 1960, pp. 490-499.

[5] B. Lewis and D.J. Berg, *Therads primer: a guide to multithreaded programming*, Prentice Hall, 1996.

[6] D.R. Morrison, PATRICA—Pratical Algorithm To Retrieve Information Coded in Alphanumeric, *Journal of the ACM*, 15, 1968, pp. 514-534.

[7] E.G. Coffman and J. Eve, File Structures Using Hashing Functions, *Communications of the ACM*, 13, 1970, pp. 427-432, 436.

[8] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, MIT Press/McGraw-Hill, 1990.

[9] H.R. Lewis and L. Deneberg, *Data Structures and their Algorithms*, Harper-Collins, 1991.