PIR WITH P-CACHE: A NEW PRIVATE INFORMATION RETRIEVAL PROTOCOL WITH IMPROVED PERFORMANCE

Dr. Abu Sayed Md. Latiful Hoque[1] and Gahangir Hossain[2]
[1]Department of Computer Science and Engineering,
Bangladesh University of Engineering and Technology (BUET), Dhaka-1000, Bangladesh.
Email: asmlatifulhoque@cse.buet.ac.bd
[2]Department of Computer Science and Engineering,
Chittagong University of Engineering and Technology (CUET), Chittagong-4349, Bangladesh.
Email: gahangir@gmail.com

ABSTRACT

The increasing model of e-commerce infrastructure opens the door for secure transaction of information over the Net, keeping some records private as users' choice within a few seconds. A client, while frequently retrieving his records, seldom wishes to hide the identity of the records to the database server. Private Information Retrieval (PIR) protocols allow users to retrieve information from a database while keeping their query private. Existing protocols have pros and cons in terms of communication and computational complexity. In some PIRs the computational complexity is reduced to O(1), and optimal, but still holds the high communication complexity O(N_s) as there is a huge amount of communication needed to satisfy the query. In our research a new PIR, namely PIR with P-cache, is introduced, based on the concept of database caching technology. In an average case, whenever a client requests a query in the system, there is a high probability to satisfy it from the P-cache instead of accessing the main database server that contains a huge amount of records. Our protocol outperforms the existing protocols in terms of both communication and computational complexity for queries supporting from P-cache and hence an improved performance.

Keywords: PIR, P-cache, Database Caching, Hashing.

1.0 INTRODUCTION

As online transaction is increases day by day, the use of secured online software with database access and information retrieval is also ever-increasing. Private Information Retrieval (PIR), a protocol that allows the user to retrieve information from a database server while hiding the identity of the record retrieved by the user, meets the requirements of the next generation e-commerce applications. Some applications of the PIR are patent databases, pharmaceutical databases, media databases, digital library, e-voting system, ethical hacking, bioinformatics, secret sharing etc.

It is a big challenge to create database with efficient PIR protocols that can ensure a good or at least acceptable response time. The main costs of PIR protocols are the communication and computational cost during query processing. A number of research works have been performed in recent years to reduce both the costs. Chor et al. [1] states two fundamental assumptions for PIR. Firstly, PIR is impossible unless we consider sending the entire database to the client as a solution; that is communication complexity is proven to the total size of the database. Secondly, in order to answer a query, the entire database must be read. In this case, the computational complexity is equal to the searching time on the whole database. On the other hand, it is a violation of user privacy if some records in the server are not read by client while processing.

To solve the existing problems and to improve the performance of a PIR, the database caching technology is used in our proposed PIR with P-cache. Existing technology requires preprocessing and off-line communication to improve performance. The use of P-cache technology requires a little preprocessing and offline communication and achieves reasonable performance in most cases.

The rest of the paper is organized as follows: In section 2, a review of the research in existing PIR protocols and their pros and cons are discussed. Section 3 describes the proposed PIR with P-cache technique. Details of P-cache management and its operational policies are also described. The analysis of P-cache performance is given in Section 4. In Section 5, the experimental work that has been carried out and the discussions on the experiment are explained. Finally in Section 6, conclusion of PIR with P-cache model and future works are discussed.

2.0 LITERATURE REVIEW

The main problem of PIR is to reduce the communication complexity of PIR protocols. Following the seminal work of Chor et al. [1], sequences of improvements have been achieved. Kushilevitz et al. [2] introduced the theoretical PIR following the basic concept of Chor and generated the special types of theory for the PIR simulation. The concept is compact as all theories are associated in the server side, only the designer knows about the special theory used. Development of such cryptographic policy with the IR system is a bit harder, thus implementation in real life was not possible. Cachin et al. [3], introduced another new PIR, namely computational PIR that depends completely on computational cryptography and RSA algorithm. This PIR updates the communication complexity from $O(N_s)$ to poly logarithmic range. In this PIR, computational complexity is high $O(N_s)$, as a server has to process each record in order to process a query. Asonov et al. [4] overcomes the limitation of computational complexity and introduced the first PIR practically implemented via third party Secure Coprocessor (SC or SCOP for example IBM 4758 SC). Its major demerit is the preprocessing complexity, and thus, the communication complexity is not reduced from $O(N_s)$. The communication complexity is conditionally reduced by the PIR with preprocessing and offline communication developed by Lipma [5] which is effective for offline communication and a large number of queries. Analyzing the merits and demerits of the previous PIRs Asonov [6] introduced the Almost Optimal PIR with the SC. Although, depending on the SC this PIR reduces the complexities and Optimal in a sense but suffers another high complexity for preprocessing (quadratic in N_s).

In spite of the large amount of research work done by many researchers in the field of Private Information Retrieval [7], [8], [9], [10], [11], [12], [13] the most important question that has been raised is to reduce the communication complexity or exclude the preprocessing overload are still far from resolution. The burning problem is to understand whether PIR protocols involving constant number of servers require polynomial (or poly logarithmic) amount of communication. Harder et al [14] provides the guidelines of designing database caching and Buhmann and Klein [15] examined the performance of a database cache. Young and Yung [16] explained the implementation techniques of a PIR protocol in their writing "*Cryptovirology"*.

3.0 PIR WITH P-CACHE ARCHITECTURE

The word P-cache stands for private cache and is simply a secure cache. It is a database cache that includes the cryptographic policy. In database caching techniques, P-cache stores recently accessed query results in order to satisfy the same future requests effectively, in the shortest time possible. The P-cache account holds the index value of the recently accessed queries of a particular client protected by the client's private key. Detailed of the P-cache structure using hash index and its performance analysis are explained later on.

This section describes the detail analytical model of PIR with P-cache and its basic components, system architecture, working principle, and complexities. The system is designed for single server environment.

3.1 Proposed System Architecture

The main ideas underlying this concept are the following. A user who searches for a record he already retrieved, has to repeat the search procedure and solve the same problems he encountered in recent access. From the concept of database caching, it is observed that the recently accessed queries from the main database are stored in the cache to improve the retrieving performance. This research introduces a caching technology and analyzes its performance improvement. A functional block diagram of P-cache with database server is shown in Fig.1.

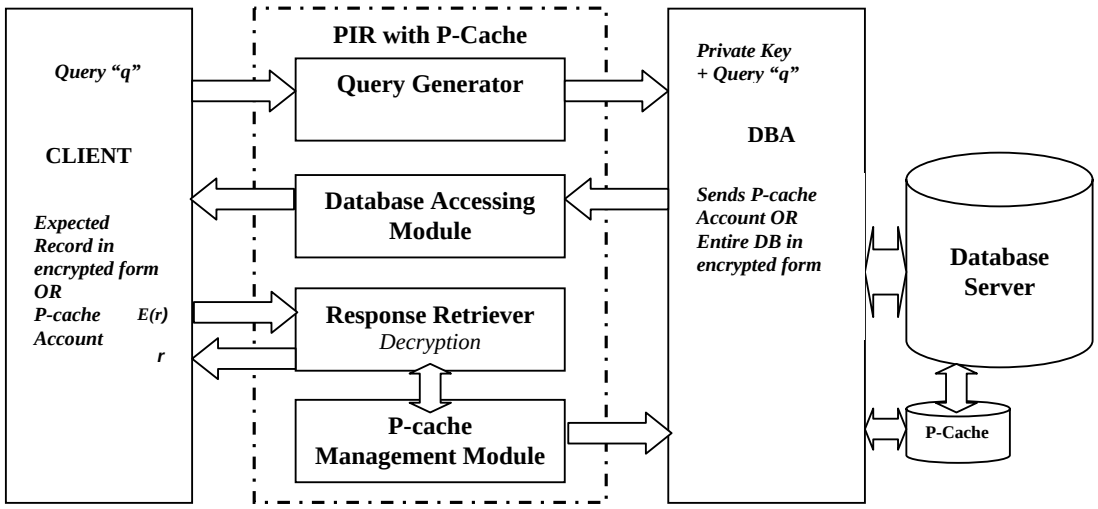
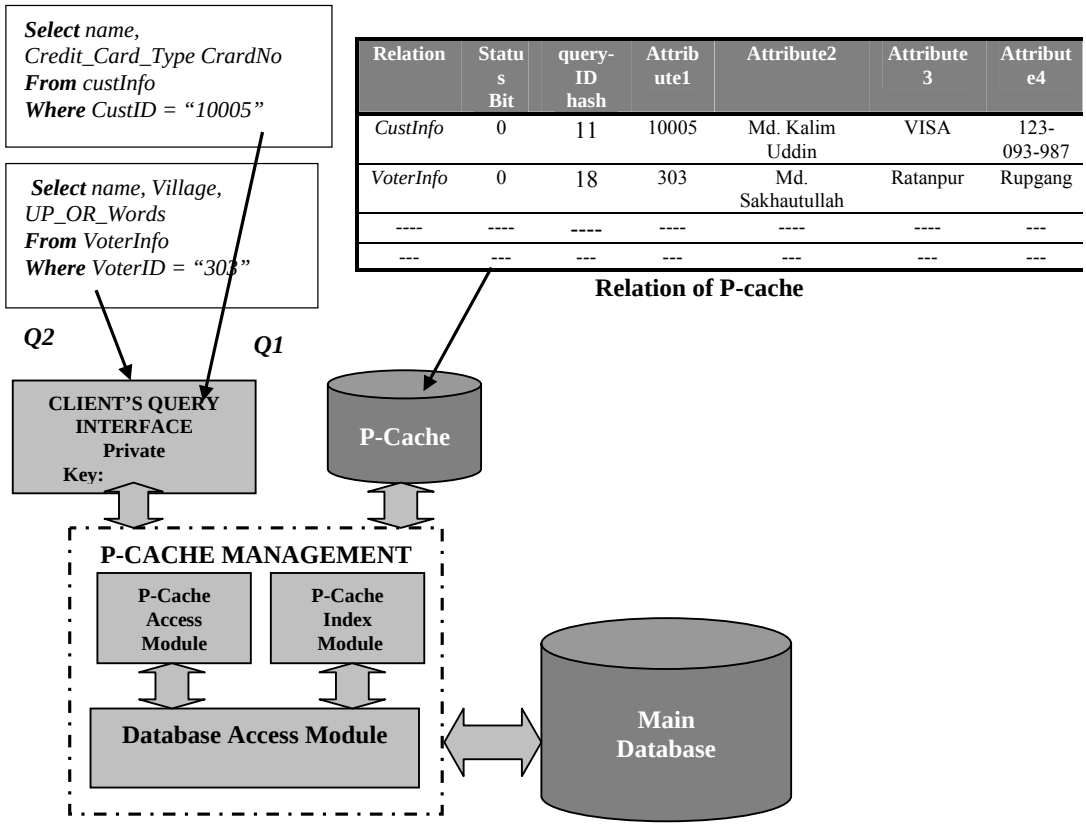


Fig.1: Functional block diagram of PIR with P-Cache

According to this protocol, before participating in the retrieving process, a client must create a valid account called P-cache account that can be accessible by his private key. Whenever the client retrieves first record from the database server the system index it within the cache using the hash key indexing. In retrieving second or next records the client applies query in his own P-cache account without reentering his private key. If the searching is not successful in P-cache, the client applies the algorithm of PIR with preprocessing and offline communication as shown in the algorithm below. After a certain period the complexity for retrieving any record reduces dramatically, occurring majority hits and a few minor misses in the cache account searches.

The main modules in PIR with P-cache protocol are the query generator module, P-cache account creation module, main database access module, response retriever module and P-cache management module as shown in the Fig.1. If the client is a valid new user, he is allowed to create a P-cache account using P-cache account creation module that accepts the private key of the client to access. Firstly, the client sends his request via query generator program. This algorithm generates the private key and corresponding query in an acceptable format to the database server. The database administrator returns the P-cache account of the corresponding registered client to his site by the main database access module. If the client is not satisfied he has an option to apply his SQL operation in the database server using the module. The requested record is then sent to the client. The client then directly retrieves it using his private key.

The P-cache Index module is mainly to hash indexing the P-cache. The LRU algorithm is also used for updating the query index. The Internal Architecture of a P-cache is shown in Fig.2.



Relation	Status Bit	query-ID hash	Attribute1	Attribute2	Attribute 3	Attribute4
CustInfo	0	11	10005	Md. Kalim Uddin	VISA	123-093-987
VoterInfo	0	18	303	Md. Sakhautullah	Ratanpur	Rupgang
----	----	----	----	----	----	---
---	---	---	---	---	---	---

Relation of P-cache

Fig.2: P-cache architecture and management

3.2 Detailed Description of P-cache Data Structure

The general structure of P-cache is an indexed structure that contains the indexes in hashed form of the relation with the attributes tagged as $Attribute_1$, $Attribute_2$…$Attribute_n$ as well as QueryID and status bit specified in the Fig.2. QueryID stores the hash index value of the corresponding query. This update operation in the P-cache depends on the status of the status bit indicated from the main database. Status bit 0 indicates that the query result is valid and no database update has been performed. The database and the P-cache are synchronized such that any update in the database changes the corresponding status bit in the P-cache from 0 to 1. The details of the P-cache management module are given in Fig.3.

Algorithm 1 : PIR with P-Cache

Inputs: Query Q for expected record R_i, Request for P-cache account (if it is created earlier)

Output: P-cache account with expected records of CLIENT's access.

Begin:

1. Requests for Cache account to the SERVER :
   IF there is no account:
   Call **P-Cache_Account_Creation ()**
   Return; // for Creating a new cache account.
2. SERVER Checks the Validity and sends CLIENT's P-

Algorithm 4 : Query_Generator

Input: *Query from the CLIENT side.*

Output: *Private Key and Query in encrypted form to the database SERVER*

Query_Generator()

Begin:

1. Initialization: PrivateKey ← New_Key;
i←1 // query index

2. *Call* RSA any cryptographic algorithm for Private Key generation.

3. **FOR** i=1 to N **DO**

Begin

Cache account.

3. Use Private Key and accesses the account by **Query_Generator()**

Apply different Queries into the P-cache Account:

3.1 FOR each Query **DO**

Begin:

3.1.1. IF the Query is successful. //**Best Case**

Call **P-Cache_Management()** and **Return.**

// Update the P-Cache.

3.1.2. ELSE *Call* **Main_Database_Access ()**;

Return.

End.

4. *Call* **Response_Retriever()**

//CLIENT retrieves his record. **Worst Case**

Call **P-Cache_Management()** and **Return.**

End.

Algorithm 2: P-Cache_Account_Creation

Input: Request for P-cache Account**.**

Output: P-cache account protected with private keys and password.

Begin:

1. Ask the SERVER to checks CLIENT's validity.

2. SERVER allows the valid new user to make his account.

3. CLIENT assigns Username and Password or any other cryptographic options to create his account and send it to the SERVER.

4. The system applies the Cryptographic logic to hide the identity.

SERVER just holds the accounts.

End.

Algorithm 3 : Main_Database_Access

// PIR with preprocessing and offline communication

Input: *Users Query and Private Key*

Output: *Encrypted Record*

Main_Database_Access()

Begin:

1. Checks the Private Key.

// homomorphic encryption or other encryption technique

2. Encrypts the whole database.

3. Send the whole encrypted database (once) to CLIENT's access.

4. CLIENT Search for the record in the encrypted hash indexed database.

End.

Consider $Query_i$ to be processed.

Use RSA Algorithm. // Private Key

I=i+1;

End.

4. Return the encrypted Unicode value to the server.

End.

Algorithm 5 : P-Cache_Management

Input: *P-Cache with initially* $\boldsymbol{n_p = 0}$ *index. //* n_p *= Number of instantaneous records in P-cache.* $\boldsymbol{N_{max}}$ *= Maximum allowable records for the P-Cache.*

Output: *Hash indexed and Updated P-Cache.*

P-Cache_Management()

Begin:

1. Initialization: $n_p \leftarrow 0$;

2. FOR n_p *!=* N_{max} **DO**

Begin:

2.1 *Apply* hash indexing to index corresponding database

record retrieved by the CLIENT.

2.2 Put the index values in a circular queue structure.

End.

3. *Apply* LRU (Least Recently Used Algorithm) in the queue for update.

End.

Algorithm 6 : Response_Retriever

Input: *The Encrypted Record* $\boldsymbol{E(R_i)}$**.**

Output: *Clients Expected record.*

Response_Retriever()

Begin:

1. CLIENT sends the private key and encrypted record.

2. Apply the symmetric cryptographic logic of decryption.

$\mathbf{D(E\ (R_i))} = \mathbf{R_i}$ // SERVER removes his encryption.

3. Send the appropriate record to the CLIENT.

End.

Fig.3: Algorithm of P-cache management

3.2.1 Internal Structure and Operations

P-cache data structure contains a set of hash indexed buckets, the unit of storage, typically a disk block, that can store one or more records. Let k denotes the set of all search key values, and let B denotes the set of all bucket addresses. A hash function $h(K_i)$ is a function from K_i to B. To insert a record with search key K_i, we compute $h(K_i)$, then search the bucket with the address. In case of hash collision, the values are put in the same bucket. The maximum number of same hash values in a particular bucket can occupy is implemented by hash chaining to reduce the collision. In case of bucket overflow, reserved buckets are used to store the additional records. It is analyzed that sometime the reserved buckets may not contain sufficient room to allow the new values inserted by the user, in which case, the replacement policy is used.

In order to query the P-cache, we perform a lookup on search key value K_i. We simply compute $h(K_i)$, then search the corresponding bucket with the address. In case of update, we compute the hash value of the record to be updated $h(K_i)$, then search the corresponding bucket for that record and update the record from the bucket.

We consider two types of clustering: user oriented clustering and query oriented clustering. In user oriented clustering, buckets are accessed by the user ID. This structure is more secured and no user can access the encrypted data of others despite having some overheads for indexing redundant records. Different users may perform the same query. The records are redundant in the P-cache hence the utilization of the P-cache is reduced. On the other hand, query oriented structure, that is proposed in this research (as shown in Fig.4) has better storage utilization of PIR protocol.

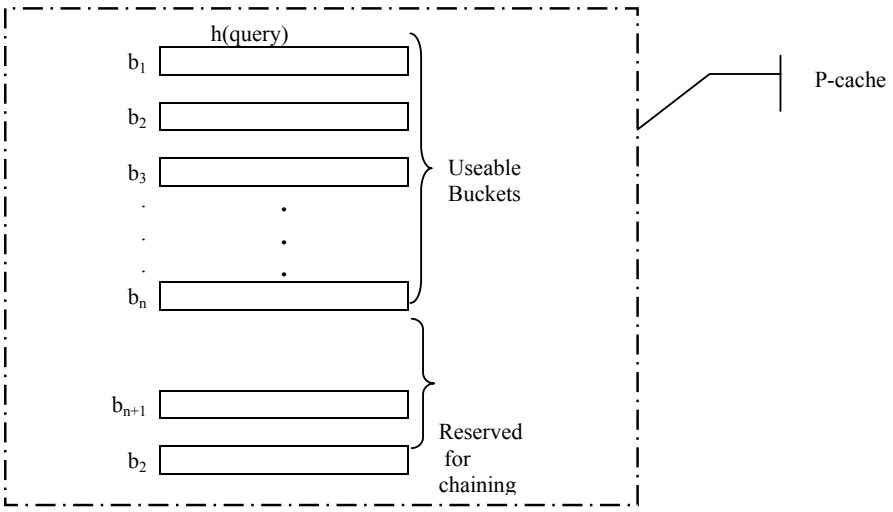


Fig.4: Single table hash file organized with clustering on query.

Here, b_i stands for bucket of records in the cache. A sample hash (query) in the *VoterInfo* schema hash(Select * , From *VoterInfo*, where *VoterId* = "AAA";) --> b_i. Let us consider, the total number of users participating in the retrieving process is N and total number of queries are Q. So the cache size $S_p \equiv K \times N \times Q$ where K is a constant. The size of a bucket in the P-cache is S_b and P-cache size is S_p. Then total number of buckets n_b is calculated as, $n_b = \dfrac{p - cache\ size\ (S_p)}{bucket\ size(S_b)}$. Again let us consider the average number of records for i^{th} query is r_{qi} and the record size is S_r. Thus, the space required for i^{th} query = $r_{qi} \times S_r$ So, the caching size required for m number of queries is, $S_p = S_r \times \left(\sum_{i=1}^{m_q} r_{qi} \right)$.

Again, let us consider the size of a record is S_r, the numbers of queries for user i is n_{ui}, the numbers of records returned for j^{th} query of user i = r_j. Thus, the cache size for i^{th} user = $S_r\left(\sum_{i=1}^{n_{ui}} r_j\right)$ and the total cache size S_{pu} = $S_r \sum_{i=1}^{n}\left(\sum_{i=1}^{n_{ui}} r_j\right)$

3.2.2 Algorithm for Bucket management in P-cache

In order to design an algorithm for bucket management, let us consider bi as the i^{th} bucket for allocation, h_{max} is the maximum allowable value, r_{qi} is the number of records returned in query q_i, S_r is the size of a record, S_b be the size of a bucket, $S_{used}(q_i)$ be the used space of query q_i, $S_{unused}(q_i)$ is the unused space of query q_i, $S_{reserved}$ is the space reserved for allocation, br is the reserved bucket for allocation and b_{empty} is the empty buckets with sufficient unused space. The algorithm *Bucket_used_space_calculation* in Fig.5 calculates the used space in the bucket. The used space in a bucket is calculated by the size of a record multiplying with total records stored in the bucket for the corresponding queries. Then the unused or free space of the corresponding bucket can be calculated by subtracting the total size used by the queries from the full bucket size. Finally, the total unused space in the cache is calculated by summing up all empty spaces in the buckets used by different users. In order to insert a query record in the p-cache, the records are first encrypted by an encryption algorithm. The status bit of corresponding query is then reset. This is also shown by another algorithm namely *Insert query_result* with *Insert_record* module in Fig. 5. The hash value is inserted as a queryID into the main relation of the cache in order to establish the relationship with cache index structure. Insertion of queries in the bucket is also explained by another algorithm. The Algorithm *Query_unique* calculates the number of unique queries in the cache.

Algorithm: Bucket_used_space_calculation
Bucket_used_space_calculation (S_b, r_i)
Begin
S_{ui} ← empty
For each record in the query q_i do
For each attribute in the record r_i do
S_a ← The size used by each attribute in a record.
// calculating allocated records size
S_{ri} ← S_{ri} + S_a // S_{ri} is the size of the record r_i
S_{ui} ← S_{ui} + S_{ri} // used space in the bucket
//S_b (unused) ← S_b – S_{ui} // used space in the bucket
Return S_b (used)
End

Algorithm: Insert query_result
Insert query_result(u_i)
Begin
bi ← hash(u_i)
Result ← set of records (randomly generated)
For i=1 to r_i
Call Insert_record (u_i, j)
End

Procedure: Insert_record (u_i, j)
Begin
Encrypt_record(u_i,j) // Applying encryption algorithm
Reset status bit
query_ID ← hash(query)
Add query_ID
//put encrypted attribute value to corresponding location.
End

Algorithm: Query_unique
Query_unique (q_i, UserID)
Begin
For each user by UserID in the p-cache do
For each query qi of user j do
queryID ← qi // Assigns a unique id for each query.
Index the queries in the Binary Search.
// to eliminate the redundant queries.
End for
End for
Count (Number of elements in the BST) // counts the total number of unique queries.
End

Fig.5: Algorithm for bucket management

3.2.3 Calculating P-cache Utilization

The P-cache utilization is calculated as the ratio of total used space with respect to total space of the p-cache.

Utilization = Total used space/ Total space = $\frac{\sum_{i=1}^{n_p} S_{pu}}{S_p}$.

Let us consider in a certain period, the number of useable bucket is = P_u%. The number of reserved bucket is therefore = (100-P_u) %. The effective utilization is then experimented by randomly choosing values to reserve the buckets. It is observed that a useable space of 80% or more supports better utilization. 80% for reserve bucket acts as the threshold value for the utilization. A graph, utilization versus P_u shows the significance of a threshold value selection for the P-cache structure.

A structure of a P-cache is considered as the arrangement of useable and reserved buckets that stores number of records returned for queries q_1, q_2, q_3….q_{ui} indexed by hash(query) is shown in Fig.6. hash(query)→ i^{th} bucket: represents the hash values are stored in the i^{th} buckets accordingly.

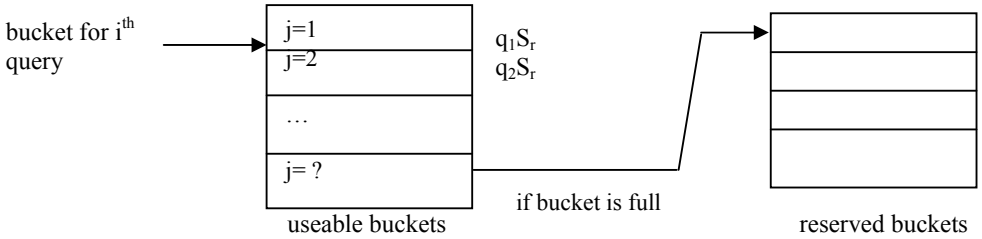


Fig.6: Structure of a block with useable and reserved buckets.

4.0 ANALYSIS OF P-CACHE PERFORMANCE

P-cache performance is analyzed calculating the utilization of memory space and query redundancy in an experimental data set.

4.1 Complexity Analysis of the Protocol

The complexity analysis of the protocol specifies its applicability in real life. By definition, a communication complexity $O(N_s)$ means only N_s records must be communicated between Server and the Client. There is a huge increase in the amount of communication needed for first PIR protocol. The user or client needs to communicate $O(N_s)$ bits instead of $O(\log N_s)$ in the usual non-private scheme. But in case of PIR with P-cache, there is a minor dependency for a query to transfer data from main database server to cache as the user can retrieve his expected records from the cache directly. The communication complexity is the number of data transferred between database server and client for retrieving a query. It seems to be unique and optimal using the cache concept as the majority of the queries is satisfied from the cache instead of database server. Computational Complexity is the complexity caused for processing records in order to answer the query. In PIR with P-cache the complexity are analyzed as in two cases:

Case-1: Query is satisfied from P-Cache.

A small number of records namely $n_p << N_s$. has to be processed by the system. So the computational complexity is $O(n_p)$ which is near to $O(1)$. As the P-cache is hash-indexed and all queries are satisfied from cache, the communication complexity is $O(1)$.

Case-2: Query is not satisfied from P-Cache.

The communication complexity is equal to $O(n_p) + O(N_s) = O(N_s)$. The computational complexity is therefore $O(n_p+N_s)$ as the system has to process all records in the cache and the main database server.

5. RESULTS AND DISCUSSIONS

This section discusses the experimental setup for the simulation and the corresponding results.

5.1 Experimental Setup and Query response time calculation

The PIR with P-cache is implemented over C++ compiler running on 1.7 GHz Pentium IV processor with 512MB memory. The operating system is Microsoft XP. The dataset is stored in Microsoft SQL Server 2000 on the same hardware configuration. The protocol is tested in randomly generated three schemas (***StdInfo*** joined with ***books*** in Digital Library System, ***CustInfo*** joined with ***products*** in PIR based Electronic Payment System, and ***VoterInfo*** in e-Voting System). In case of different schemas in main database and P-cache the performance of the protocol is recorded as like Table 1. The bar diagrams (Fig.7) shows the comparative performance of the two protocols for two specific queries. Whether the query is satisfied from P-cache it is shown in Fig.8. The operation is performed in two queries Q_1 and Q_2:

Q_1: hash(**Select** name, Credit_Card_Type, CardNo; **From** custInfo;**Where** CustID = "10005")
Q_2: hash(**Select** name, Village, UP_OR_Words; **From** VoterInfo;**Where** VoterID = "303")

Table1: Query response time for different protocols along with PIR with P-Cache

Query	Query Response Time (ms)			
	Computational PIR	**PIR Preprocessing and off-line communication**	**Almost Optimal**	**PIR with P-Cache Query satisfied from P-Cache**
Q_1	23220	3812	2023	3823
Q_2	32920	2294.30	3214	2295
Q_1	23221	3811	2032	0.0102
Q_2	32920	2294	3215	0.011
Q_1	24228	3812	2031	0.009
Q_2	32924	2294	3216	0.013

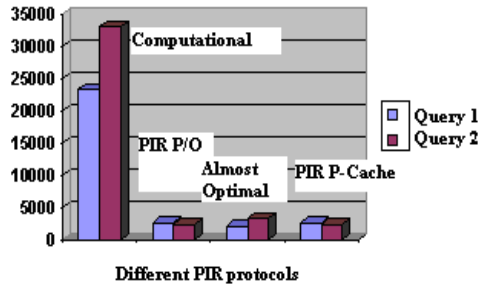


Fig.7: Query response time comparison for different PIR protocols.

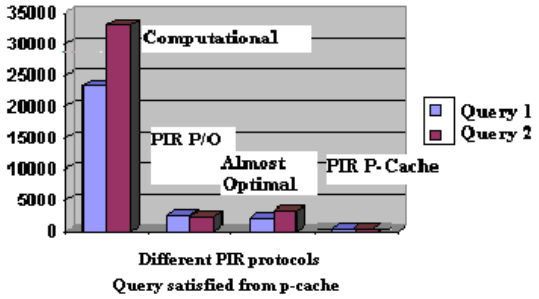


Fig.8: Query response time comparison for different PIR protocols. Query satisfied from p-cache

5.2 P-cache utilization for *hash(query)* indexing

Total utilization of the cache can be calculated by merging the used spaces in the buckets of the cache divided by its total space. In the experiment the numbers of useable buckets varied from 90% to 60% and the number of reserved buckets varied from 10% to 40% accordingly. It shows relatively better performance then the *hash(userID)* utilization as there is no chance for the same query to be redundant in *hash(query)* indexing. It is also impossible for a userID to be redundant. Thus the query oriented clustering of the P-cache performs better in the case of memory utilization. Finally the graph in Fig.9 shows cache utilization of the *hash(query)* indexing. In Fig.10 number of unique queries versus query redundancy is shown.

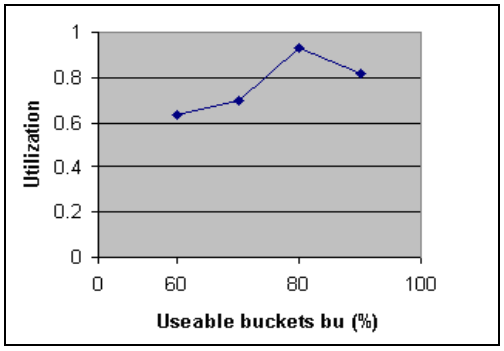


Fig.9: P-cache utilization for *hash(q_i)* indexing

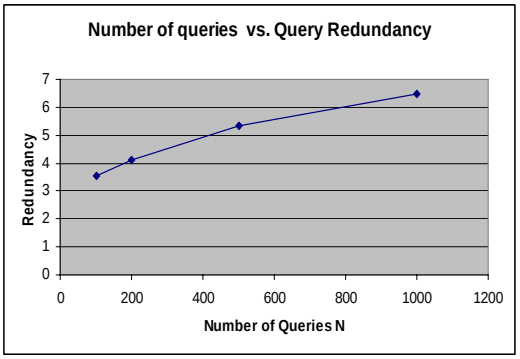


Fig.10: Number of unique queries vs. redundancy

In Table.2, complexities of different PIR protocols with the proposed "PIR with P-cache" are summarized. The comparison shows that PIR with P-cache performs better for large number of redundant queries. In worse case, whenever the record is not satisfied by the P-cache, the proposed system has to use PIR with preprocessing and offline communication to retrieve the expected record from the main database server.

Table.2: Pros and Cons in terms of complexities of PIR protocols with PIR with P-cache.

PIRs	Pros	Cons
Theoretical PIR	Compact concept	Relatively hard to develop
Computational PIR	C_{comm}= polynomial time and updated to poly logarithmic time. No Pre processing	$C_{comp} = O(Ns)$.
Hardware Based PIR	$C_{comm} = O(1)$ No Pre-Processing	$C_{comp} = O(N_s)$.
PIR with preprocessing and offline communication	$C_{comp} = O(1)$	$C_{comm} = O(N_s)$ Pre-processing required
Almost Optimal PIR	$C_{comp} = O(1)$ $C_{comm} = O(1)$ *Assumed*	Pre-processing High. Complexity Quadratic in N
PIR with P-cache	$C_{comp} = O(1)$ $C_{comm} = O(1)$ *for app. 80%* queries	Pre-processing required for app. 20% queries

6.0 CONCLUSION AND FURTHER RESEARCH

We have presented a new PIR protocol with database caching technique, namely P-cache that significantly improves the performance in terms of computational and communication complexities over the existing PIR protocols. The computational complexity improves from $O(N_s)$ to $O(1)$ and the communication complexity improves from $O(N_s)$ to $O(n_s)$ in average case, where, N_s and n_s are the size of the main database server and the size of the P-cache in terms of total number of records respectively.

We have considered two types of clustering for P-cache structure: user oriented and query oriented. It is experimentally justified that query oriented clustering supports better utilization than the user oriented clustering as the user oriented clustering suffers from data redundancy in the P-cache.

In PIR with P-cache, as each user is assigned a separate cache account with cryptographic policy, it ensures a new isolated security system that is independent of the database server. Hence, the system hides the identity of the user or the records retrieved by him to the database server perfectly.

The protocol is independent of preprocessing in average case. But in the worst case, there is a minor dependency of the main database access. Hence, the computational complexity rises to $O(N_s+n_s) \approx O(N_s)$ and the communication complexity also rises to $O(N_s)+O(n_s) \approx O(N_s)$.

This work is simulated on a centralized system. Future work is aimed for multi server (k-server) or web application server in distributed environment.

REFERENCES

[1] B. Chor, O. Goldreich, E. Kushelevitz and M. Sudan, "Private Information Retrieval," *Journal of the ACM,* Vol. 45, pp. 965-982, 1998.

[2] E. Kushilevitz and R. Ostrovsky, "Replication is Not Needed: Single Database Private Information Retrieval," *In 38th Annual Symposium of Foundations of Computer Science,* IEEE Computer Society, Miami Branch, Florida, pp. 364-373, October 20-22, 1997.

[3] C. Cachin, S. Micali, and M. Stadler, "Computationally Private Information Retrieval with Polylogarithmic communication," *in Advances in Cryptology- EUROCRIPT'99,* Vol. 1592, Prague, Czech Republic. Springer Verlag, pp. 402-414, May 1999.

[4] D. Asonov, J. Christoph and J. C. Freytag, "Private Information Retrieval," *in the proceedings of the 1st Workshop on Privacy Enhancing Technologies* San Francisco, USA, pp. 26-37, April 2001.

[5] H. Lipmaa and S. Laur, "On Security of Sub-linear Oblivious Transfer," *in Technical Report 2004*, International Association for Cryptologic Research, Vol. 63, pp. 10-35, February 25, 2004.

[6] D. Asonov, J. Christoph and J. C. Freytag, "Almost Optimal Private Information Retrieval," *in the proceedings of the 2nd Workshop on Privacy Enhancing Technologies* San Francisco, USA, pp. 56-72, April 2002.

[7] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala, "Using Probabilistic I/O Automata to Analyze an Oblivious Transfer Protocol," in *CSAIL*, MIT, Vol. 1001, pp. 12-24, 2005.

[8] H. Lipmaa, "An Oblivious Transfer Protocol With Log-Squared Communication," *in the 8th Information Security Conference*, Singapore, Vol. 3650, pp. 314-328, September 20-23, 2005.

[9] G. Semeraro, F. Abbattista, N. Fanizzi and S. Ferilli, "Intelligent Information Retrieval in a Digital Library Service," in *COGITO Project IST*-1999-13347, pp. 123-146, March 2003.

[10] D.Woodruff and S.Yekhanin, "A Geometric Approach to Information Theoretic Private Information Retrieval," *in Proc. of the 43rd IEEE Symposium on Foundation of Computer Science,* Berlin, pp. 261-10, June 2004.

[11] S.W. Smith and D.Safford, "Practical Private Information Retrieval with Secure Co-processor," *in Technical report, IBM research Division,* T.J. Watson Research Center, pp. 26-39, July 2000.

[12] A. Iliev and S. Smith, "Private Information Storage with Logarithmic Space Secure Hardware," *in Advances in Cryptology EUROCRIPT05,* pp. 55-68, December 2005.

[13] S. W. Smith and D.Safford, "Practical Server Privacy with Secure Co-processor," in *IBM Systems Journal,* Vol. 40, no. 3, pp. 11-23, 2005.

[14] T. Harder and A. Buhman, "Value Complete, Column Complete, Predicate Complete-Magic words Driving the design of Cache groups", in *The VLDB Journal*, Vol. DOI 10.10007/S1007/s 00778-006, pp. 0035-0039, January 2007.

[15] A. Buhmann and J. Klein, "Examining the Performance of a Constrain-Based Database Cache," in *ADBIS*, Varna, F.J. Curry House of scientist, Bulgaria, 2007.

[16] A.L. Young and M.M. Yung, "An Implementation of Tagged Private Information Retrieval," in *Chapter-8, Cryptovirology,* Ver. 2.1, 2005-2006.

BIOGRAPHY

Dr. Abu Sayed Md. Latiful Hoque received his PhD in the field of Computer & Information Science from University of Strathclyde, Glasgow, UK in 2003 with Commonwealth Academic Staff Award. He obtained M.Sc. in Computer Science & Engineering and B.Sc. in Electrical & Electronic Engineering from Bangladesh University of Engineering & Technology (BUET) in 1997 and 1986 respectively. He has been working as a faculty member in the Department of Computer Science & Engineering at BUET since 1990 and currently his position is an Associate Professor. He is a Fellow of Institute of Engineers Bangladesh (IEB) and Bangladesh Computer Society. His research interest includes Data Warehouse, Data Mining, Information Retrieval and Compression in Database Systems.

Gahangir Hossain received his B.Sc. Engineering in Electronics and Computer Science from Shahjalal University of Science & Technology (SUST), Sylhet, Bangladesh in 2000, and completed his M.Sc. Engineering in Computer Science & Engineering in 2008 from Bangladesh University of Engineering & Technology (BUET), Bangladesh. He is a member of Institute of Engineers Bangladesh (IEB) and currently working as an Assistant Professor of Chittagong University of Engineering & Technology, Chittagong, in Computer Science & Engineering Department. His research interest includes Human Computer Interaction, Information Retrieval, Software Engineering, Machine Learning, Natural Language Processing and Bioinformatics.