Research Article **ISSN: 2395 – 5597**

International Journal of Engineering and Robot Technology

Journal home page: www.ijerobot.com

ACCES PATTERN USING DATABASE INFORMATION NOT PRESENT IN CACHE

D. Rajesh*[1] and D. Ramesh[2]

*[1]Department of Computer Science Engineering, Universal College of Engineering and Technology, Tamilnadu, India.

ABSTRACT

Private Information Retrieval (PIR) is one of the fundamental security requirements for database outsourcing. A major threat is information hacking form database access patterns generated by query executions used by the data base server. The standard private information retrieval schemes which are widely regarded as theoretical solutions, entail o(n) computational overhead per query for a data base with n items. Recent works propose to protect access patterns by introducing a trusted component with constant storage size. The resulting privacy assurance is a strong as private information retrieval (PIR), through with o(1) online computation cost, they still have o(n) amortized cost per query due to periodically full database shuffles. In this paper, we design a novel scheme in the same model with provable security, which only shuffles a portion of the database without storage, the amortized server computational complexity is reduced than previous algorithm. Our scheme can protect the access pattern privacy of database of billions of entries, at lower cost.

KEY WORDS

PIR- Private Information Retrieval, ISP-Internet Service Provider and DNS-Domain Name System.

Author of correspondence:

D. Rajesh,
Department of Computer Science Engineering,
Universal College of Engineering and Technology,
Tamilnadu, India.

Email: rajeshd936@gmail.com

INTRODUCTION[1]

Cache Memory

A cache is a component that transparently stores data so that future requests for that data can be served faster. The data that is stored within a cache might be values that have been computed earlier or duplicates of original values that are stored elsewhere. If requested data is contained in the cache (cache hit), this request can be served by simply reading the cache, which is comparatively faster. Otherwise (cache miss), the data has to be recomputed or fetched from its original storage location, which is comparatively slower.

Operations of cache

Hardware implements cache as a block of memory for temporary storage of data likely to be used again. CPUs and hard drives frequently use a cache, as do web browsers and web servers.

A cache is made up of a pool of entries. Each entry has a datum (a nugget (piece) of data) - a copy of the same datum in some backing store. Each entry also has a tag, which specifies the identity of the datum in the backing store of which the entry is a copy.

When the cache client (a CPU, web browser, operating system) needs to access a datum presumed to exist in the backing store, it first checks the cache. If an entry can be found with a tag matching that of the desired datum, the datum in the entry is used instead. This situation is known as a cache hit. So, for example, a web browser program might check its local cache on disk to see if it has a local copy of the contents of a web page at a particular URL. In this example, the URL is the tag, and the contents of the web page are the datum. The percentage of accesses that result in cache hits is known as the hit rate or hit ratio of the cache[1].

The alternative situation, when the cache is consulted and found not to contain a datum with the desired tag, has become known as a cache miss. The previously uncached datum fetched from the backing store during miss handling is usually copied into the cache, ready for the next access.

During a cache miss, the CPU usually ejects some other entry in order to make room for the previously uncached datum. The heuristic used to select the entry to eject is known as the replacement policy. One popular replacement policy, "least recently used" (LRU), replaces the least recently used entry (see cache algorithm). More efficient caches compute use frequency against the size of the stored contents, as well as the latencies and throughputs for both the cache and the backing store[2]. This works well for larger amounts of data, longer latencies and slower throughputs, such as experienced with a hard drive and the Internet, but is not efficient for use with a CPU cache.

Database caching

Many applications today are being developed and deployed on multi-tier environments that involve browser-based clients, web application servers and backend databases. These applications need to generate web pages on-demand by talking to backend databases because of their dynamic nature, making middle-tier database caching an effective approach to achieve high scalability and performance. In three tier architecture, application tier and data tier will be in different hosts. Throughput of the application is affected by the network speed. This network overhead shall be avoided by having database at the application tier. As commercial databases are heavy weight, it is not practically feasible to have application and database at the same host. There is lot of light-weight databases available in the market, which shall be used to cache the data from the commercial databases.

APPLICATIONS OF CACHE

CPU cache

Small memories on or close to the CPU can operate faster than the much larger main memory. Most CPUs since the 1980s have used one or more caches, and modern high-end embedded, desktop and server microprocessors may have as many as half a dozen, each specialized for a specific function. Examples of caches with a specific function are the D-cache and I-cache (data cache and instruction cache).

Disk cache

While CPU caches are generally managed entirely by hardware, a variety of software manages other caches. The page cache in main memory, which is an example of disk cache, is managed by the operating system kernel. While the hard drive's hardware disk buffer is sometimes misleadingly referred to as "disk cache", its main functions are written sequencing and read perfecting. Repeated cache hits are relatively rare, due to the small size of the buffer in comparison to the drive's capacity. However, high-end disk controllers often have their own on-board cache of hard disk data blocks. Finally, fast local hard disk can also cache information held on even slower data storage devices, such as remote servers (web cache) or local tape drives or optical jukeboxes.

Such a scheme is the main concept of hierarchical storage management[3].

Web cache

Web browsers and web proxy servers employ web caches to store previous responses from web servers, such as web pages. Web caches reduce the amount of information that needs to be transmitted across the network, as information previously stored in the cache can often be re-used. This reduces bandwidth and processing requirements of the web server, and helps to improve responsiveness for users of the web. Web browsers employ a built-in web cache, but some internet service providers or organizations also use a caching proxy server, which is a web cache that is shared among all users of that network[3]. Another form of cache is P2P caching, where the files most sought for by peer-to-peer applications are stored in an ISP cache to accelerate P2P transfers. Similarly, decentralised equivalents exist, which allow communities to perform the same task for P2P traffic, for example, Corelli.

Other caches

The BIND DNS daemon caches a mapping of domain names to IP addresses, as does a resolver library. Write-through operation is common when operating over unreliable networks (like an Ethernet LAN), because of the enormous complexity of the coherency protocol required between multiple write-back caches when communication is unreliable. For instance, web page caches and side network caches (like those in NFS or SMB) are typically read-only or write-through specifically to keep the network protocol simple and reliable. Search engines also frequently make web pages they have indexed available from their cache. For example, Google provides a "Cached" link next to each search result. This can prove useful when web pages from a web server are temporarily or permanently inaccessible[4]. Another type of caching is storing computed results that will likely be needed again, or memorization. Cache, a program that caches the output of the compilation to speed up the second-time compilation, exemplifies this type. Database caching can substantially improve the throughput of database applications, for example in the processing of indexes, data dictionaries, and frequently used subsets of data. Distributed caching uses caches spread across different networked hosts, for example, Corelli.

The difference between buffer and cache

The terms "buffer" and "cache" are not mutually exclusive and the functions are frequently combined; however, there is a difference in intent. A buffer is a temporary memory location that is traditionally used because CPU instructions cannot directly address data stored in peripheral devices. Thus, addressable memory is used as an intermediate stage. Additionally, such a buffer may be feasible when a large block of data is assembled or disassembled (as required by a storage device), or when data may be delivered in a different order than that in which it is produced. Also, a whole buffer of data is usually transferred sequentially (for example to hard disk), so buffering itself sometimes increases transfer performance or reduces the variation or jitter of the transfer's latency as opposed to caching where the intent is to reduce the latency. These benefits are present even if the buffered data are written to the buffer once and read from the buffer once. A cache also increases transfer performance. A part of the increase similarly comes from the possibility that multiple small transfers will combine into one large block. But the main performance-gain occurs because there is a good chance that the same datum will be read from cache multiple times, or that written data will soon be read. A cache's sole purpose is to reduce accesses to the underlying slower storage. Cache is also usually an abstraction layer that is designed to be invisible from the perspective of neighboring layers[5].

PROBLEM DESCRIPTION

Database access patterns leakage information during query execution when the user is accessing the data from the database. Unnecessary access of data from the database leads to the leakage of information from the data bases. Malicious server access the data from the trusted component delays the trusted user in accessing the data. Repeated use of queries in accessing the data leads to the storage overhead in the trusted component. The storage overhead in the trusted during accessing the data from the database is reduced without using the cache storage.

EXISTING SYSTEM

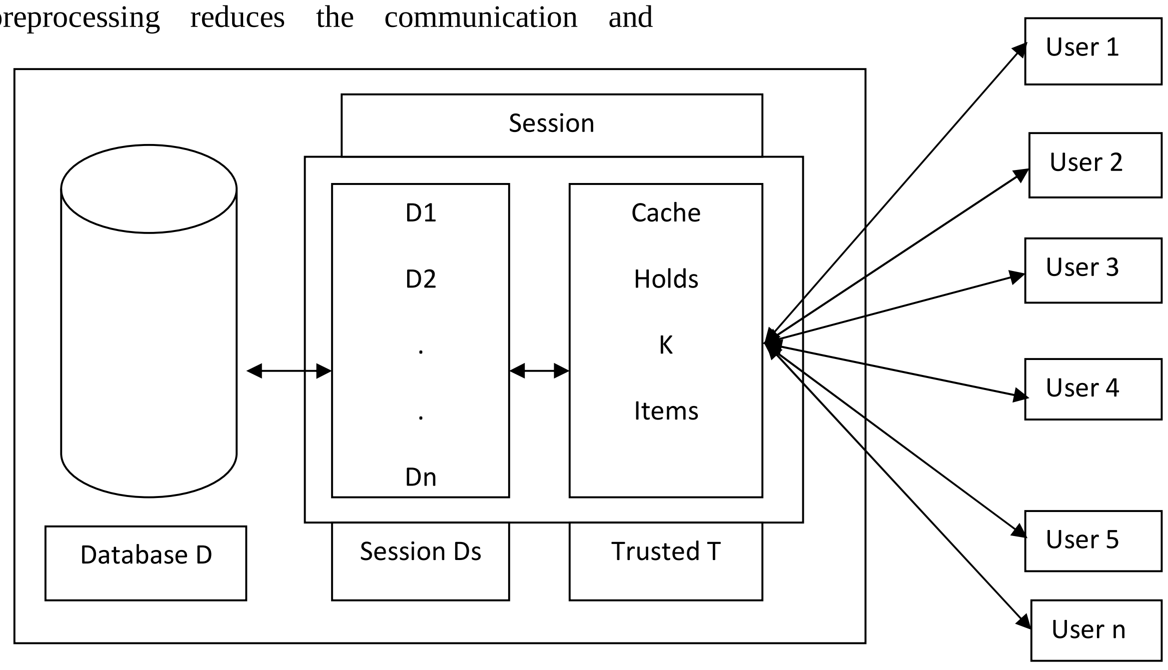
Private information retrieval (PIR) protocol allows a user to retrieve an item from a server in possession of a database without revealing which item they are retrieving. PIR is for the server to send an entire copy of the database to the user. There are two problems in PIR, one is to make the server computationally bounded and the other is there are multiple non-cooperating servers, each having a copy of the database. Length-Flexible homomorphic public key encryption technique all the users uses the same modulus for generating the key pairs. A threshold decryption protocol is used to handle messages of any length. This leads to higher computation cost and there is no support of trusted hardware component. Reducing the server computation in private information retrieval holds the server database and linear computation is performed. In order to avoid this problem PIR with preprocessing is applied. This approach before processing the queries the database server computers and stores the data as a polynomial. PIR preprocessing reduces the communication and computation cost. Not feasible to preprocess and store in the database.

PROPOSED SYSTEM

Access pattern using the database without cache storage uses line of search and a novel PIR scheme, an approach reduces the malicious database server in accessing the data, and avoids the database from full shuffles. Security is enhanced by encrypting the database and shuffling the partial database. Line of search reduces the complexity and removing the cache storage from the trusted component reduces the storage overhead.

Evaluation of Database Scheme

A database is a collection of information that is organized so that it can easily be accessed, managed, and updated. Database consists of sensitive information of the particular systems. Only authorized users can be able access the sensitive data from the database using access patterns. Unauthorized users can also access the sensitive data from the database due to insufficient security of the database system. For increasing the privacy of the database private information retrieval (PIR) schemes are implemented (Figure No.1).



Database Host (H)

Figure No.1: Architectural diagram

CONCLUSION

A novel scheme to prevent database access patterns from being exposed to a malicious server. By virtue of twin-retrieval and partial-shuffle, our scheme avoids full-database shuffle and reduces the amortized server computation complexity. Although the hierarchy-based ORAM algorithm family can protect access patterns with at most cost O(log2), they are plagued with large constants hidden in the big-O notations. With a modest cache k=1024, our construction outperforms those poly-logarithm algorithms for databases of 3*1010 entries. In addition, our scheme has much less server storage overhead. We have formally proved the scheme's security following the notion of PIR and showed our experiment results which confirm our performance analysis.

ACKNOWLEDGEMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I would like to express my gratitude towards my parents and member of Universal College of Engineering and Technology for their kind co-operation and encouragement which help me in completion of this project.

I would like to express my special gratitude and thanks to industry persons for giving me such attention and time.

My thanks and appreciations also go to my colleague in developing the project and people who have willingly helped me out with their abilities.

CONFLICT OF INTEREST

We declare that we have no conflict of interest.

BIBLIOGRAPHY

1. Arnold T W and Van Doorn L P. "The IBM PCIXCC: A new cryptographic coprocessor for the IBM eserver," *IBM J. Res. Devel*, 48, 2004, 475-487.
2. Beimel A, Ishai Y, Kushilevitz E and Raymond J F. "Breaking the O(n1/(2k-1)) barrier for information-theoretic private information retrieval, *In Proc. IEEE FOCS'*, 02, 2002, 261-270.
3. Beimel A, Ishai Y and Malkin T. "Reducing the servers computation in private information retrieval: PIR with preprocessing," *In Proc. CRYPTO'00*, 2000, 55-73.
4. Black J and Rogaway P. "Ciphers with arbitrary finite domains," *In Proc. CT-RSA*, 2002, 114-130.
5. Chor B and Gilboa N. "Computationally private information retrieval," *In Proc. STOC'97*, 29th, 1997, 304-313.