On the Access Complexity of PIR Schemes

Yiwei Zhang
Dept. of Computer Science
Technion
Haifa 3200003, Israel
ywzhang@cs.technion.ac.il

Eitan Yaakobi
Dept. of Computer Science
Technion
Haifa 3200003, Israel
yaakobi@cs.technion.ac.il

Tuvi Etzion
Dept. of Computer Science
Technion
Haifa 3200003, Israel
etzion@cs.technion.ac.il

Moshe Schwartz
Department of ECE
Ben-Gurion Univ. of the Negev
Beer Sheva 8410501, Israel
schwartz@ee.bgu.ac.il

Abstract—Private information retrieval has been reformulated in an information-theoretic perspective in recent years. The two most important parameters considered for a PIR scheme in a distributed storage system are the storage overhead and PIR rate. The complexity of the computations done by the servers for the various tasks of the distributed storage system is an important parameter in such systems which didn't get enough attention in PIR schemes. As a consequence, we take into consideration a third parameter, the *access complexity* of a PIR scheme, which characterizes the total amount of data to be accessed by the servers for responding to the queries throughout a PIR scheme. We use a general covering codes approach as the main tool for improving the access complexity. With a given amount of storage overhead, the ultimate objective is to characterize the tradeoff between the rate and access complexity of a PIR scheme. This covering codes approach raises a new interesting coding problem of generalized coverings similarly to the well-known generalized Hamming weights.

I. Introduction

Private information retrieval (PIR) protocols, first introduced by Chor, Goldreich, Kushilevitz, and Sudan in [5], allow a user to retrieve a data item from a database without revealing any information about the identity of the item to any single server. The original formulation of the PIR problem considers replicating a binary string on several non-communicating servers. The objective is to optimize the communication cost, including both the upload cost and the download cost, for privately retrieving one single bit. In recent years, the information-theoretic reformulation of the PIR problem assumes the more practical scenario in which the files are of arbitrarily large size. Under this setup, the number of uploaded bits can be neglected with respect to the corresponding number of downloaded bits since the upload does not depend on the size of the file [4]. This reformulation introduces the *rate* of a PIR scheme to be the ratio between the size of the retrieved file and the total number of downloaded bits from all servers. The supremum of achievable rates over all PIR schemes is defined as the *PIR capacity*. In their pioneering work [13] Sun and Jafar determine the exact PIR capacity of the classical PIR model of replication.

Starting from [12], the research of PIR has been combined with distributed storage system instead of the replication-based system. This brings in the other important parameter, i.e., the *storage overhead* of the distributed storage system, defined as the ratio between the total number of bits stored on all the servers and the number of bits of the database. Several papers have been studying the relation between the storage overhead and the rate of a PIR scheme. Chan et al. [4] offer a tradeoff between the storage overhead and rate for linear PIR schemes. They show that when each server stores a fraction $0 < \epsilon \leq 1$ of the database, then the rate of a linear PIR scheme should be at most $\frac{N-1/\epsilon}{N}$, where N is the number of server. Tajeddine et al. [16] propose a PIR scheme achieving this upper bound when the storage code is an arbitrary (N, K)-MDS code, so $\epsilon = \frac{1}{K}$ and the PIR rate

is $\frac{N-K}{N}$. Banawan and Ulukus [2] show that the exact PIR capacity when using an arbitrary (N, K)-MDS storage code is $(1 + \frac{K}{N} + \cdots + \frac{K^{M-1}}{N^{M-1}})^{-1}$, a value dependent on the number of files M and tends to $\frac{N-K}{N}$ when M approaches infinity. However, similar to the scheme of Sun and Jafar [13], this optimal scheme can be implemented only if the file size L is an exponential function of M [14], [18]. For a more practical setting we are more interested in the case when L is at most a polynomial value in terms of M and the scheme of Tajeddine et al. [16] works for this setup.

Recall the development of the research on distributed storage systems: Besides optimizing repair bandwidth or storage for distributed storage systems, *access complexity* is also a concern since the time of reading data may cause a bottleneck. The research of optimal-access MDS codes started in [15]. A similar idea in locally repairable codes was introduced in [9] for the sake of reducing the nodes to be accessed. The complexity of the computations done by the servers for the various tasks of the distributed storage system is an important parameter in such systems which didn't get enough attention in PIR schemes. The only work which took the computational complexity of the servers, in the new PIR model, into account, was done by Lavauzelle [11]. Our approach is completely different. For practical use of PIR protocols in distributed storage systems, we should also consider the access complexity in the scheme. However, to the best of our knowledge, the access complexity of PIR has not been studied in previous works so far. In fact, most known PIR schemes require accessing almost all of the data stored on each server in the worst case. The next example demonstrates the concepts and improvements for the access complexity that we study in this work. We will consider the worst case in this paper, but the average case is also very interesting from a theoretical and practical points of view.

Example 1. *Consider the following 2-server PIR scheme where each server stores the whole database $\boldsymbol{x} = (\boldsymbol{x}^1, \boldsymbol{x}^2, \dots, \boldsymbol{x}^M)$. A user chooses an arbitrary binary vector $\boldsymbol{a} = (a_1, \dots, a_M) \in \mathbb{F}_2^M$ and then sends $\boldsymbol{a}$ and $\boldsymbol{a} + \boldsymbol{e}_f$ to the two servers respectively. From the responses $\sum_{i=1}^M a_i \boldsymbol{x}^i$ and $\sum_{i=1}^M a_i \boldsymbol{x}^i + \boldsymbol{x}^f$ the user successfully retrieves the desired file $\boldsymbol{x}^f$ privately. While the main advantage of this solution is its low download complexity, it suffers from extremely large access complexity since in the worst case almost all M files are accessed on each server. Hence, in this scheme the bottleneck will no longer be the upload or download time, but the access time to read all files. The access complexity can be improved at the cost of increasing the storage overhead. That is, when storing more information in the servers the computation $\boldsymbol{a} \cdot \boldsymbol{x}$ will require to access a fewer number of files. For example, assume we also store in each server the file $\boldsymbol{x}_\Sigma$ given by $\boldsymbol{x}_\Sigma = \sum_{i=1}^M \boldsymbol{x}^i$. Then, in the worst case, the server will read only $M/2$ files. Thus we save half of the*

access complexity in the tradeoff of storing one additional file on each server.

Intuitively for a PIR scheme in a distributed storage system there will be a relationship among the three parameters: storage overhead, PIR rate, and access complexity. The ultimate objective is to characterize the tradeoff of any two parameters when fixing the third. In this paper, we make a first step towards solving this problem. Given the number of servers N, the number of files M, we fix the size of the storage space for each server (and thus fix the storage overhead) and analyze the rate and access complexity of several PIR schemes.

The rest of the paper is organized as follows. In Section II, we give a formal statement of the PIR problem studied in the paper. In Section III, we discuss how to improve the access complexity using covering codes. In Section IV, we analyze the rate and access complexity for several PIR schemes. Finally, Section V concludes the paper.

II. PROBLEM STATEMENT

A PIR scheme for a distributed storage system consists of the following parameters:

- The system has N servers. A database consists of M files $\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^M$ of equal length L. The size of the database is then ML.
- Each server stores ϵML bits, $\epsilon > 0$. Thus the total storage is ϵNML.
- The *storage overhead* is defined as the ratio between the total storage and the size of the database, i.e., ϵN.
- The storage code of the system is an encoding mapping $(\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^M) \in \mathbb{F}_2^{ML} \longrightarrow (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N), \ \boldsymbol{y}_n \in \mathbb{F}_2^{\epsilon ML}$.
- To retrieve a file, a user downloads ρ_n bits from the nth server. The total download cost is then $\sum_{n=1}^N \rho_n$.
- The *rate* Ω of a PIR scheme is defined as the ratio of the size of a desired file and the number of downloaded bits, i.e. $\Omega = \frac{L}{\sum_{n=1}^N \rho_n}$.
- The ρ_n downloaded bits from the nth server are functions of the data $\mathbf{y}_n$ it stores. The calculation of these downloaded bits requires the server to access $\delta_n ML$ bits in $\mathbf{y}_n$. δ_n is called the *access complexity* of the nth server. The *total access complexity* is defined as $\Delta = \sum_{n=1}^N \delta_n$.

We call a 6-tuple $(N, M, L, \Omega, \Delta, \epsilon)$ *achievable*, if for a distributed storage system with parameters N, M and L, we have a PIR scheme with rate Ω, total access complexity Δ, and each server stores a fraction $\epsilon > 0$ of the whole database (and thus the storage overhead is ϵN). When N, M and L are clear from the context or not relevant, we abbreviate the 6-tuple as a 3-tuple $(\Omega, \Delta, \epsilon)$. The ultimate objective is to characterize the exact tradeoff between any two of the parameters Ω, Δ and ϵ when fixing the third. In this paper we make a first step towards solving this problem by finding some achievable 3-tuples of $(\Omega, \Delta, \epsilon)$ with a predetermined ϵ.

Intuitively the storage space can be divided into two parts. One part represents the indispensable storage for a particular PIR scheme and is referred to as *the storage for PIR*. This part represents the independent symbols stored on each server. The remaining part is jointly designed with the former part for improving the access complexity on each server. We illustrate this idea via the following example.

Example 2. *Consider a distributed storage system storing a database containing M files $\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^M$ of equal size L. Assume we have $N = 3$ servers with $\epsilon = 1$, i.e., each server can store ML bits. A user wants to retrieve a specific file $\boldsymbol{x}^f$.*

One way is to allocate all the storage space to be used for PIR, so each server stores the whole database. Divide each file into two equal parts $\boldsymbol{x}^m = (\boldsymbol{x}_1^m, \boldsymbol{x}_2^m)$. A user chooses two independent random vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ in $\mathbb{F}_2^M$. He asks for $\sum_{i=1}^M a_i \boldsymbol{x}_1^i + \sum_{i=1}^M b_i \boldsymbol{x}_2^i$, $\sum_{i=1}^M a_i \boldsymbol{x}_1^i + \sum_{i=1}^M b_i \boldsymbol{x}_2^i + \boldsymbol{x}_1^f$ and $\sum_{i=1}^M a_i \boldsymbol{x}_1^i + \sum_{i=1}^M b_i \boldsymbol{x}_2^i + \boldsymbol{x}_2^f$ from the three servers respectively. Therefore he downloads $\frac{3L}{2}$ bits, so the rate of the scheme will be $\Omega = 2/3$. Each server will access almost all the data in the worst case. Altogether almost $3ML$ bits should be accessed throughout the scheme. Then the total access complexity will be $\Delta = 3$. So we have an achievable 3-tuple $(\Omega = 2/3, \Delta = 3, \epsilon = 1)$.

Yet another way is to only use half of the storage for PIR and the other half for improving the access complexity. Let each server store only half of the database. Say we have $\{\boldsymbol{x}_1^m : 1 \le m \le M\}$ on the first server, $\{\boldsymbol{x}_2^m : 1 \le m \le M\}$ on the second server and a coded form $\{\boldsymbol{x}_1^m + \boldsymbol{x}_2^m : 1 \le m \le M\}$ on the third server. Again a user chooses two independent random vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ in $\mathbb{F}_2^M$. He makes two queries from each server and gets the responses as follows:

Server I	Server II	Server III
$\sum_{i=1}^M a_i \boldsymbol{x}_1^i + \boldsymbol{x}_1^f$	$\sum_{i=1}^M a_i \boldsymbol{x}_2^i$	$\sum_{i=1}^M a_i (\boldsymbol{x}_1^i + \boldsymbol{x}_2^i)$
$\sum_{i=1}^M b_i \boldsymbol{x}_1^i$	$\sum_{i=1}^M b_i \boldsymbol{x}_2^i + \boldsymbol{x}_2^f$	$\sum_{i=1}^M b_i (\boldsymbol{x}_1^i + \boldsymbol{x}_2^i)$

This is exactly the scheme of Tajeddine et al. in [16] when using a $(3, 2)$-MDS storage code. In this scheme the download will be $3L$ bits so the rate will be $\Omega = 1/3$. To improve the access complexity, each server stores a coded form of the data using a covering code approach instead of storing $\{\boldsymbol{x}_1^m : 1 \le m \le M\}$, $\{\boldsymbol{x}_2^m : 1 \le m \le M\}$ or $\{\boldsymbol{x}_1^m + \boldsymbol{x}_2^m : 1 \le m \le M\}$ in their original form. For each query a server will only need to read about $0.22ML$ bits (to be explained in Section III). So altogether at most $1.32ML$ bits are accessed in the scheme, resulting in the total access complexity $\Delta = 1.32$. So we have an achievable 3-tuple $(\Omega = 1/3, \Delta = 1.32, \epsilon = 1)$.

III. ACCESS COMPLEXITY USING COVERING CODES

A (binary) *covering code* $\mathcal{C}$ of length ℓ with *covering radius* R is a set of vectors in $\{0, 1\}^\ell$ such that for every vector $\boldsymbol{u} \in \{0, 1\}^\ell$ there exists a codeword $\boldsymbol{c} \in \mathcal{C}$ with Hamming distance $d_H(\boldsymbol{c}, \boldsymbol{u}) \le R$. Covering codes were extensively studied and comprehensive information on them can be found in [7]. For linear covering codes this property can be translated as follows.

Proposition 3. [8] *Let $\mathcal{C}$ be a linear code of length ℓ, dimension k, redundancy $r = \ell - k$, and a parity check matrix $\mathcal{H}$ of size $r \times \ell$. Then, $\mathcal{C}$ is a covering code with covering radius R if and only if for every column vector $\boldsymbol{s} \in \{0, 1\}^r$ there exists a row vector $\boldsymbol{y} \in \{0, 1\}^\ell$ of Hamming weight at most R, such that $\mathcal{H} \cdot \boldsymbol{y}^T = \boldsymbol{s}$.*

The other way to explain the covering radius of a linear code is as follows. A column vector $\boldsymbol{s} \in \{0, 1\}^r$ is actually a syndrome corresponding to a particular coset of the code $\mathcal{C}$ in $\mathbb{F}_2^\ell$. In this coset one can find a vector $\boldsymbol{y} \in \mathbb{F}_2^\ell$ (not necessarily unique) with minimum Hamming weight. The vector $\boldsymbol{y}$ is known as a *coset leader* and its weight is known as the *coset weight*. Then one can get the vector $\boldsymbol{s}$ by summing up the columns of $\mathcal{H}$ indexed by the support set of $\boldsymbol{y}$. Thus the covering radius of a linear code is exactly the maximum of all its coset weights. Linear covering codes can be used to improve the access complexity as follows.

Suppose we have a database $\boldsymbol{x}$ which can viewed as a $t \times r$ matrix, i.e. $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_r)$, where each $\boldsymbol{x}_i$, $1 \le i \le r$, is a column vector of length t. Let $\mathcal{C}$ be a linear code of length ℓ,

dimension k, redundancy $r = \ell - k$, covering radius R and an $r \times \ell$ parity check matrix $\mathcal{H} = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_\ell]$. Each server stores the database $\boldsymbol{x}$ encoded by the columns of $\mathcal{H}$. That is, the server stores ℓ column vectors $\boldsymbol{z}_i = \boldsymbol{x} \cdot \boldsymbol{h}_i$ for $1 \leq i \leq \ell$. In other words, $\boldsymbol{z}_i$ is a linear combination of the files (column vectors) of the database $\boldsymbol{x}$, with coefficients taken from $\boldsymbol{h}_i$. The user who chooses an arbitrary binary column vector $\boldsymbol{s} = (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_r)^T$, for the query, to the j-th server, wants to retrieve from the j-th server the vector $\boldsymbol{x} \cdot \boldsymbol{s}$. To compute $\boldsymbol{x} \cdot \boldsymbol{s}$, the server first finds the coset leader $\boldsymbol{y}$ such that $\mathcal{H} \cdot \boldsymbol{y}^T = \boldsymbol{s}$. Then computing $\boldsymbol{x} \cdot \boldsymbol{s}$ is equivalent to

$$\boldsymbol{x} \cdot \boldsymbol{s} = \boldsymbol{x} \cdot (\mathcal{H} \cdot \boldsymbol{y}^T) = \boldsymbol{x} \cdot \left(\sum_{i:y_i=1} \boldsymbol{h}_i \right) = \sum_{i:y_i=1} \boldsymbol{z}_i.$$

Since the Hamming weight of the coset leader $\boldsymbol{y}$ is at most R, it follows that we only need to access at most R columns of $\mathcal{H}$ to compute $\boldsymbol{x} \cdot \boldsymbol{s}$. Moreover, $\mathcal{H}$ can be chosen in the form $\mathcal{H} = [I_r \mid A_{r \times (\ell-r)}]$ and thus we can always have a systematic form of the original data. The asymptotic connection between the length ℓ, covering radius R, and the dimension k of the linear covering code can be roughly estimated by the sphere-covering bound

$$2^k \cdot 2^{H(R/\ell)\ell} \approx 2^\ell,$$

or

$$\frac{k}{\ell} + H(R/\ell) = 1,$$

so $H(R/\ell) = 1 - \frac{k}{\ell} = \frac{r}{\ell}$, where $H(\cdot)$ is the binary entropy function. By setting the covering radius to be $R = \alpha r$ and the size of the storage $\ell = \beta r$, we have

$$H\left(\frac{\alpha}{\beta} \right) = \frac{1}{\beta}. \tag{1}$$

Solving this equation and the relation between α and β can be represented as a function $\alpha = f(\beta)$, depicted in Fig. 1.
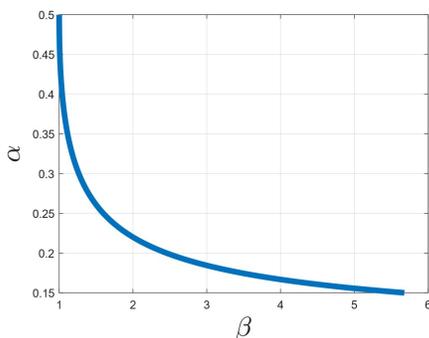
Fig. 1: Access vs. Storage

It is not an easy task to explicitly construct linear codes achieving the above sphere-covering bound. However, the existence of such codes has been proved:

Proposition 4. [6] *Let $0 \leq R \leq \ell/2$. Then there exists an infinite sequence of linear codes C_ℓ of growing length ℓ with covering radius $R(C_\ell) \to R$ and the code rate is between $1 - H(R/\ell)$ and $1 - H(R/\ell) + O(\ell^{-1} \log_2 \ell)$.*

In a PIR scheme which does not take access complexity into consideration, the data stored on a server can be usually represented as r independent strings of the same length and a query asks for a linear combination of these strings. Using the covering code approach above, we may store a coded

version of the data instead of only storing the original form. The access complexity could be improved as follows.

Theorem 5. *Suppose there exists a linear binary covering code with redundancy r, code length βr and covering radius αr. Given a set of r independent strings $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_r\}$, a server can store a coded form of these strings as $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{\beta r}\}$, such that computing any linear combination of $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_r\}$ requires only accessing at most αr substrings in $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{\beta r}\}$. The asymptotic relation of α and β is $H(\frac{\alpha}{\beta}) = \frac{1}{\beta}$.*

A further remark is that adding redundancies in the storage does not affect the privacy of the original scheme. In essence the privacy is only related to the set of queries. Finally, we note that the problem of reducing the access complexity when replying to queries of the form mentioned in this section is not relevant only for PIR schemes. The approach can be relevant for other models which require this or similar computation. An example for such a problem was studied in [10] for the partial-sum problem where the authors also used covering codes. However, since the computations involved integer numbers, the storage overhead was exponential with the number of items.

IV. PIR RATE VS. ACCESS COMPLEXITY

Now, we analyze the PIR rate and access complexity for two kinds of PIR schemes which provide good tradeoff between the rate, access complexity, and storage overhead. The schemes by Tajeddine et al. [16] and a scheme of Blackburn, Etzion and Paterson (B-E-P scheme) [3] are considered. Given ϵ indicating the size of the storage space of each server, we first choose some $0 \leq \pi \leq \epsilon$ indicating that the size of the storage for PIR, i.e., the amount of storage of independent symbols. Using this π fraction of storage we implement a proper PIR scheme with high rate. Then we analyze the total access complexity of this scheme by making use of the remaining $\epsilon - \pi$ fraction of the storage space.

A. The scheme of Tajeddine et al. [16]

When $\pi = \frac{1}{K}$, $K < N$, the rate of the scheme of Tajeddine et al. [16] achieves the upper bound $\Omega = \frac{N-K}{N}$ proposed by Chan et al. in [4]. So we begin with analyzing how to improve the access complexity of this scheme using the covering code approach.

Recall the framework of the scheme. Let each file $\boldsymbol{x}^m \in \mathbb{F}_2^L$ be represented in the form of a matrix $\mathbf{X}^m = \{\boldsymbol{x}_{i,j}^m : 1 \leq i \leq N - K, 1 \leq j \leq K\}$, where each $\boldsymbol{x}_{i,j}^m$ represents a binary substring of length $\frac{L}{K(N-K)}$. Let $\Lambda = [\lambda_1, \ldots, \lambda_N]$ be a $K \times N$ generator matrix of the storage code. Then the nth server stores $\mathbf{X}^m \lambda_n$, which are $N - K$ linearly independent substrings as functions of $\boldsymbol{x}^m$. The whole storage on the nth server is thus a concatenation of altogether $M(N - K)$ linearly independent substrings. Each server will receive K queries, where each query asks for a certain linear combination of these $M(N - K)$ substrings.

We make use of the additional storage of size $(\epsilon - \frac{1}{K})ML$ bits on each server. Select a covering code with redundancy $r = M(N - K)$ with code length βr where $\beta = K\epsilon$. Instead of storing the r substrings of $\boldsymbol{y}_n$ in their original form, the server stores βr substrings according to the covering code approach. Then by Theorem 5, to answer each query, the server only needs to access at most $\alpha r = f(\beta)r$ substrings. Recall that each server receives K queries. Thus each server will access at most $\min\{f(\beta)rK, r\}$ substrings, since accessing the r linearly independent substrings are already enough for

computing any linear combination. Therefore, the total number of bits accessed by each server is $\min\{f(K\epsilon)ML, \frac{ML}{K}\}$ and thus the total access complexity over all servers will be $\Delta = \min\{Nf(K\epsilon), \frac{N}{K}\}$.

For example, select $N = 10$. Let $\epsilon = 1$. We can take arbitrary $1 \leq K \leq 9$ and apply the scheme of Tajeddine et al. The PIR rate and total access complexity are listed as follows. $\Delta' = \frac{N}{K}$ corresponds to the total access complexity when there is no redundancy in each server.

K	$\Omega = \frac{N-K}{N}$	Δ	Δ'
1	0.9	5.000	10.000
2	0.8	2.201	5.000
3	0.7	1.845	3.333
4	0.6	1.668	2.500
5	0.5	1.556	2.000
6	0.4	1.477	1.667
7	0.3	1.418	1.429
8	0.2	1.250	1.250
9	0.1	1.111	1.111

The table above indicates the covering code approach does improve the total access complexity for $K \neq 8, 9$. We further note that Tajeddine et al. mention that their scheme could be implemented with cutting each file into $\frac{\text{l.c.m.}(K,N-K)}{K} \times K$ substrings instead of $(N - K) \times K$, and correspondingly the number of subqueries for each server is $\frac{\text{l.c.m.}(K,N-K)}{N-K}$ instead of K. This modification allows us to further improve the access complexity. Select a covering code with redundancy $r' = M\frac{\text{l.c.m.}(K,N-K)}{K}$ with code length $\beta r'$ where $\beta = K\epsilon$. Then by Theorem 5 each query will access at most $\alpha r' = f(\beta)r'$ substrings. Each server receives $\frac{\text{l.c.m.}(K,N-K)}{N-K}$ queries and thus the number of bits to be accessed on each server is at most $f(\beta)r'\frac{\text{l.c.m.}(K,N-K)}{N-K} \times \frac{L}{K \times \frac{\text{l.c.m.}(K,N-K)}{K}} = f(K\epsilon)ML\frac{\text{l.c.m.}(K,N-K)}{K(N-K)}$. Therefore when K and $N - K$ are not coprime, the total access complexity will be further improved as $\Delta = Nf(K\epsilon)\frac{\text{l.c.m.}(K,N-K)}{K(N-K)} = \frac{Nf(K\epsilon)}{\gcd(K,N-K)}$. Thus some of the results in the example above can be improved as follows.

K	$\Omega = \frac{N-K}{N}$	Δ
2	0.8	1.100
4	0.6	0.834
5	0.5	0.311
6	0.4	0.739
8	0.2	0.685

In conclusion, applying the scheme of Tajeddine et al. results in several achievable 3-tuples as follows.

Theorem 6. *In a distributed storage system consisting of N servers, for every $1 \leq K < N$, $\epsilon \geq \frac{1}{K}$, the tuple $(\Omega = \frac{N-K}{N}, \Delta = \min\{\frac{Nf(K\epsilon)}{\gcd(K,N-K)}, N/K\}, \epsilon)$ is achievable.*

We close this subsection by discussing the possibility of further improving the access complexity for the scheme of Tajeddine et al. Note that each server may receive multiple queries. The data accessed by a server when responding to different queries may have certain overlap. Reconsider Example 1, if we have two queries, then the server may read only $\frac{3M}{4}$ files instead of reading $\frac{M}{2}$ files twice. Further improving the access complexity for the scheme of Tajeddine et al. (by taking advantage of possible overlap when reading multiple queries) relies on a good solution to the coding theoretic problem presented in the next subsection.

B. Generalized coset weights

Given a binary linear code, for every τ cosets of the code, choose one vector from each coset and find the size of the union of their support sets. The minimum of this value is called the τ-*coset weight* of these τ cosets. What is the maximum value of all τ-coset weights? When $\tau = 1$ this is the covering radius R of the linear code. When $\tau \geq 2$, we would like to see weights smaller than τR.

The τ-coset weights (for covering) are akin to the generalized Hamming weights (for distance) defined in [17] which were considered in hundreds of papers.

Let $[n, k, d]$ code denote a binary linear code of length n, dimension k, and minimum Hamming distance d.

Lemma 7. *The τ-coset weight of a code $\mathcal{C}$ is the minimum number of columns ℓ in the parity check matrix $\mathcal{H}$ of $\mathcal{C}$, such that for each τ syndromes of $\mathcal{C}$, there exists a set of ℓ columns of $\mathcal{H}$ which has τ linear combinations of this set to form these τ syndromes.*

Theorem 8. *The τ-coset weight of an $[n, k, d]$ code $\mathcal{C}$ is at most $n - k$ for each $\tau \geq 1$.*

Proof. The parity check matrix $\mathcal{H}$ of $\mathcal{C}$ has $n - k$ linearly independent columns. A set of such $n - k$ columns covers a word in each coset of $\mathcal{C}$. □

Theorem 9. *The τ-coset weight of the $[2^m - 1, 2^m - 1 - m, 3]$ Hamming weight is τ for each $1 \leq \tau \leq m$.*

Theorem 10. *The τ-coset weight of the $[2^m, 2^m - 1 - m, 4]$ extended Hamming weight is $\tau + 1$ for each $1 \leq \tau \leq m$.*

For many types of BCH codes with minimum distance d and covering radius R we have proved that the 2-coset weight is smaller than $2R$. This was generalized in some cases for τ-coset weights with $\tau > 2$. This and other related results will be considered in the full version of this paper.

C. Using several parallel B-E-P schemes

Consider the scheme of Tajeddine et al. when $K = 1$, i.e., replicated databases. Each file $\boldsymbol{x}^m \in \mathbb{F}_2^L$ is divided into $N-1$ substrings $\boldsymbol{x}_1^m, \ldots, \boldsymbol{x}_{N-1}^m$ of length $\frac{L}{N-1}$. Each server stores $\boldsymbol{y} = \{\boldsymbol{x}_1^1, \ldots, \boldsymbol{x}_{N-1}^1, \boldsymbol{x}_1^2, \ldots, \boldsymbol{x}_{N-1}^2, \ldots, \boldsymbol{x}_1^M, \ldots, \boldsymbol{x}_{N-1}^M\}$, altogether $(N - 1)M$ substrings. A user chooses a random binary vector $\boldsymbol{v}$ of length $(N-1)M$. The Nth server receives the query vector $\boldsymbol{v}$ and the nth server receives the query vector $\boldsymbol{v} + \boldsymbol{e}_{(f-1)(N-1)+n}$, $1 \leq n \leq N - 1$, where f is the index of the desired file. Then from the response of the nth server and the Nth server, the user retrieves the string $\boldsymbol{x}_n^f$, $1 \leq n \leq N - 1$.

The B-E-P scheme recently proposed by Blackburn, Etzion and Paterson suggests a different way, whose original motivation is to optimize the upload complexity of the query vectors. A user who wants to retrieve the file $\boldsymbol{x}^f$ chooses M elements $z_1, \ldots, z_M \in \mathbb{Z}_N$ uniformly and independently at random. The nth server receives $(b_{1n}, \ldots, b_{Mn})$ where $b_{fn} = z_f + n$ (mod N) and $b_{mn} = z_m$ for $m \neq f$ and then responds with $\sum_{m=1}^{M} \boldsymbol{x}_{b_{mn}}^m$, where $\boldsymbol{x}_0^m$ represents the all-zero vector.

The main difference is that a query in the former scheme asks for an arbitrary linear combination of all the $(N - 1)M$ substrings while a query in the latter scheme asks for a linear combination with a restricted pattern, i.e., at most one substring from each file is involved in the linear combination. This restriction may allow for a better way to improve the access complexity than the covering code approach.

Example 11. $N = 3$, $M = 3$. *Consider the necessary amount of storage overhead for a PIR scheme with rate 2/3 and total access complexity* 1. *The scheme of Tajeddine et al. gives an achievable tuple* $(2/3, 1, 13/6)$, *which applies a covering code of length 13, redundancy 6 and covering radius 2 to store the six substrings* $\{\boldsymbol{x}_1^1, \boldsymbol{x}_2^1, \boldsymbol{x}_1^2, \boldsymbol{x}_2^2, \boldsymbol{x}_1^3, \boldsymbol{x}_2^3\}$. $\epsilon = 13/6$ *cannot be improved for the scheme of Tajeddine et al. since 13 is the minimum length of a linear covering code with redundancy 6 and covering radius 2 [7, p. 202]. However, if we use the B-E-P scheme, then each server can store the following 11 substrings:* $\{\boldsymbol{x}_1^1, \boldsymbol{x}_2^1, \boldsymbol{x}_1^2, \boldsymbol{x}_2^2, \boldsymbol{x}_1^3, \boldsymbol{x}_2^3, \boldsymbol{x}_1^1 + \boldsymbol{x}_2^2, \boldsymbol{x}_1^2 + \boldsymbol{x}_2^3, \boldsymbol{x}_1^3 + \boldsymbol{x}_2^1, \boldsymbol{x}_1^1 + \boldsymbol{x}_1^2 + \boldsymbol{x}_1^3, \boldsymbol{x}_2^1 + \boldsymbol{x}_2^2 + \boldsymbol{x}_2^3\}$. *This already guarantees that we only need to read at most two substrings for any query in the B-E-P scheme. Thus the B-E-P scheme gives an achievable tuple (2/3,1,11/6).*

In the former two subsections, on each server $\frac{1}{K}ML$ bits are allocated as the storage for PIR and the remaining $(\epsilon - \frac{1}{K})ML$ bits are designed for improving the access complexity. Now consider the case when $\frac{p}{q}ML$ bits are allocated for the PIR scheme and the remaining $(\epsilon - \frac{p}{q})ML$ bits are used for improving the access complexity, where $\frac{1}{N} \leq \frac{p}{q} \leq \epsilon$ cannot be simplified to the form $\frac{1}{K}$. Once we have a proper PIR scheme with good rate in this setup, the idea for improving the access complexity will be exactly the same approach aforementioned.

As shown by [4], such a PIR scheme will have rate at most $\frac{N - \frac{q}{p}}{N}$. This model was then named as the storage constrained PIR and bounds on the capacity were considered in [1]. Particularly, when further restricting the $\frac{p}{q}ML$ bits of storage to be uncoded, [1] determined the exact capacity which is achieved by a memory sharing method plus the capacity-achieving schemes of [13]. Since the B-E-P scheme has asymptotically optimal rate, it is natural to consider the memory sharing method using several parallel B-E-P schemes.

Suppose that the file size is $L = t\ell$ and we divide each file $\boldsymbol{x}^m$ into t parts of equal size ℓ, $\{\boldsymbol{x}_1^m, \dots, \boldsymbol{x}_t^m\}$. We choose some d parts from each file and consider them as a new subdatabase, say $\{\boldsymbol{x}_j^m : 1 \leq m \leq M, 1 \leq j \leq d\}$. Then we may perform a B-E-P subscheme for this subdatabase on some $d+1$ servers. This subscheme occupies $\frac{d}{t}ML$ bits on each of the $d+1$ servers involved and contributes $\frac{d+1}{t}L$ bits to the download cost. A combined PIR scheme by the memory sharing method, is done by just dividing the database into several subdatabases and then implementing several parallel B-E-P subschemes, each on a certain subset of servers. Note that a sufficiently large t and a proper way to allocate servers for each subscheme (say, by permutations) will guarantee that each server stores exactly $\frac{p}{q}ML$ bits. Since in this scheme each server has uncoded storage, then as suggested by [1], to achieve the asymptotically optimal rate, each subscheme should be implemented on either $\lceil \frac{Np}{q} \rceil$ or $\lfloor \frac{Np}{q} \rfloor$ servers.

The rate of this scheme can be calculated as follows. Altogether a proportion η of the database is involved in subschemes on $\lceil \frac{Np}{q} \rceil$ servers and the rest proportion $1 - \eta$ is involved in subschemes on $\lfloor \frac{Np}{q} \rfloor$ servers, where $\eta \lceil \frac{Np}{q} \rceil + (1 - \eta) \lfloor \frac{Np}{q} \rfloor = Np/q$. The total download is then

$$L \cdot \left(\eta \frac{\lceil \frac{Np}{q} \rceil}{\lceil \frac{Np}{q} \rceil - 1} + (1 - \eta) \frac{\lfloor \frac{Np}{q} \rfloor}{\lfloor \frac{Np}{q} \rfloor - 1} \right).$$

Finally, the rest $(\epsilon - \frac{p}{q})ML$ bits on each server are used for improving the access complexity via the covering code approach aforementioned.

Theorem 12. *In a distributed storage system consisting of N servers, for every rational number $\frac{1}{N} \leq \frac{p}{q} \leq \epsilon$, the tuple $(\Omega, \frac{Np}{q} f(\epsilon \frac{q}{p}), \epsilon)$ is achievable, where*

$$\Omega = \left(\eta \frac{\lceil \frac{Np}{q} \rceil}{\lceil \frac{Np}{q} \rceil - 1} + (1 - \eta) \frac{\lfloor \frac{Np}{q} \rfloor}{\lfloor \frac{Np}{q} \rfloor - 1} \right)^{-1}.$$

V. Conclusion

In this paper we took into consideration the access complexity of a PIR scheme. PIR schemes with low access complexity reduce the amount of data to be accessed throughout a PIR scheme and are therefore suitable for practical use. A few methods were considered, especially ones which use covering codes. Some of these codes were applied on known schemes. It should be noted that these methods are not useful for all the known schemes, e.g. the one of Sun and Jafar [13]. Finally, the problem of generalized coset weights, which will be helpful when there are multiple queries on each server, has independent interest in coding theory.

References

[1] M.A. Attia, D. Kumar, and R. Tandon, "The capacity of private information retrieval from uncoded storage constrained databases," arXiv:1805.04104v2, May. 2018.

[2] K. Banawan and S. Ulukus, "The capacity of private information retrieval from coded databases," *IEEE Trans. on Inform. Theory*, vol. 64, no. 3, pp. 1945–1956, Mar. 2018.

[3] S. Blackburn, T. Etzion, and M. Paterson, "PIR schemes with small download complexity and low storage requirements," arXiv:1609.07027v3, Nov. 2017.

[4] T.H. Chan, S.W. Ho, and H. Yamamoto, "Private information retrieval for coded storage," *Proc. IEEE Int. Symp. on Inf. Theory*, pp. 2842–2846, Hong Kong, Jun. 2015.

[5] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *J. ACM*, vol. 45, no. 6, pp. 965–981, Nov. 1998. Earlier version in FOCS 95.

[6] G. Cohen and P. Frankl, "Good coverings of Hamming spaces with spheres", *Discrete Math.*, vol. 56, no. 2-3, pp. 125–131, Oct. 1985.

[7] G. Cohen, I. Honkala, S. Litsyn, and A. Lobstein, "Covering codes," Elsevier, 1997.

[8] G. Cohen, M. Karpovsky, H. Mattson, and J. Schatz, "Covering radius: Survey and recent results," *IEEE Trans. on Inform. Theory*, vol. 31, no. 3, pp. 328–343, May 1985.

[9] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of codeword symbols," *IEEE Trans. on Inform. Theory*, vol. 58, no. 11, pp. 6925–6934, Nov. 2012.

[10] C.-T. Ho, J. Bruck, and R. Agrawal, "Partial-sum queries in OLAP data cubes using covering codes," *IEEE Trans. on Comp.*, vol. 47, no. 12, pp. 1326–1340, Dec. 1998.

[11] J. Lavauzelle, "Private information retrieval from transversal designs," *IEEE Trans. on Inform. Theory*, to appear.

[12] N.B. Shah, K.V. Rashmi, and K. Ramchandran, "One extra bit of download ensures perfectly private information retrieval," *Proc. IEEE Int. Symp. on Inform. Theory*, pp. 856–890, Honolulu, HI, Jul. 2014.

[13] H. Sun and S.A. Jafar, "The capacity of private information retrieval," *IEEE Trans. on Inform. Theory*, vol. 63, pp. 4075–4088, July 2017.

[14] H. Sun and S.A. Jafar, "Optimal download cost of private information retrieval for arbitrary message length," *IEEE Trans. Inform. Forensics and Security*, vol. 12, pp. 2920–2932, Dec. 2017.

[15] I. Tamo, Z. Wang, and J. Bruck, "Access versus bandwidth in codes for storage," *IEEE Trans. on Inform. Theory*, vol. 60, pp. 2028–2037, Apr. 2014.

[16] R. Tajeddine, O.W. Gnilke, and S. El Rouayheb, "Private information retrieval from MDS coded data in distributed storage systems," *IEEE Trans. on Inform. Theory*, vol. 64, pp. 7081–7093, Nov. 2018.

[17] V.K. Wei, "Generalized Hamming weights for linear codes," *IEEE Trans. on Inform. Theory*, vol. 37, pp. 1412–1418, Sep. 1991.

[18] J. Xu and Z. Zhang, "On sub-packetization of capacity-achieving PIR schemes for MDS coded databases," arXiv:1712.02466, Dec. 2017.