PIRMAP: Efficient Private Information Retrieval for MapReduce (Short Paper)

Travis Mayberry   Erik-Oliver Blass   Agnes Hui Chan
College of Computer and Information Science
Northeastern University, Boston MA 02115
{travism|blass|ahchan}@ccs.neu.edu

ABSTRACT

Private Information Retrieval (PIR) allows for retrieval of bits from a database in a way that hides a user's access pattern from the server. However, its practicality in a cloud computing setting has recently been questioned. In such a setting, PIR's enormous computation and communication overhead is expected to outweigh any cost saving advantages of cloud computing. This paper presents PIRMAP, a practical, highly efficient protocol for PIR in MapReduce, a widely supported cloud computing API. PIRMAP focuses especially on the retrieval of large files from the cloud, where it achieves optimal communication complexity ($O(l)$ for retrieval of an l bit file) with query times significantly faster than previous schemes. To achieve this, PIRMAP arranges files so parallel evaluation can be done during the "Map" phase of MapReduce and aggregation can be carried out via an efficient additively homomorphic encryption scheme in the "Reduce" phase. PIRMAP has been implemented and tested in Amazon's public cloud with total database sizes of up to 1 TByte. Our performance evaluations show that PIRMAP is more than one order of magnitude cheaper and faster than "trivial PIR" on Amazon and adds only 20% overhead to a theoretical optimal PIR.

1. INTRODUCTION

Cloud computing has recently been identified as an important strategic technology [6], as it offers the advantage of greater flexibility and potentially reduced costs to companies outsourcing their data and computation. The cost advantage of cloud computing comes from the fact that cloud users do not need to maintain their own, expensive data center, but instead can pay a cloud provider for hosting. However, despite the hype, hesitations remain among major organizations. Lack of security and privacy guarantees has been identified as a major adoption obstacle for both large enterprise [17] and governmental organizations [11].

Privacy problems stem from the fact that, when a cloud user hands over his information to the cloud, he is relinquishing control of that data to a third party. Public clouds are threatened by hackers due to the much larger target they present. Insiders such as data center administrative staff can also easily access private data that has been outsourced. As multiple cloud users are hosted on the same data center ("multi tenancy"), even other cloud users might try to illegally access tenancy"), even other legitimate cloud users might try to illegally access data. Finally, as cloud providers place data centers abroad in countries with unclear privacy laws, local authorities become an additional threat to outsourced data. Such attacks are realistic and have already been reported [7, 18, 21]. This shows that the user needs to take special precautions to ensure the privacy and integrity of their cloud data.

In this paper, we envision an application scenario where a cloud user stores large amounts of data (files) with a cloud provider and subsequently wishes to retrieve a subset of those files. Such a scenario is realistic; for example, many hospitals are currently looking to take advantage of cloud storage by outsourcing patients' sensitive health records [19]. One can imagine that doctors upload their patients' health records (files) to a public cloud provider and occasionally retrieve records later on. In the face of an untrusted cloud provider, the privacy of patients has to be protected.

While encryption of data at rest helps to protect data confidentiality, this is often not sufficient: the subsequent data access patterns can leak information about patients. For example, if the outsourced data in question contains encrypted patient records, the cloud provider might learn that a patient has been diagnosed with cancer when it sees that patient's records have been retrieved by an oncologist.

It is very hard to assess what information may be leaked this way and what inferences an adversary may make. Private Information Retrieval (PIR) offers a solution to the problem of hiding access patterns [4, 8], independent of an encryption mechanism. In PIR, a database stores n strings each of size l bits, and a user can query for one l bit string without leaking *which* string to the database. The challenge is to perform retrieval of the l bit string in a more efficient manner than the trivial solution with $O(l \cdot n)$ complexity that sends the entire database.

Because of the significant overhead, it has recently been questioned whether PIR will ever become *practical* in a real-world cloud computing setting. Cloud providers such as Amazon charge their customers for both data transfer *and* CPU hours [1]. Due to the large constants involved in more advanced PIR protocols, it has been argued that trivial PIR (retrieving the whole $(l \cdot n)$ bit database) is not only faster, but also *cheaper* for the cloud customer compared to a PIR query that involves lengthy computation and data transfer [3, 13, 16].

Another open question is how to perform PIR in a real-world cloud computing environment. One of the biggest challenges in cloud computing is writing application code so it can scale easily to the large distributed systems which are characteristic in such a setting. In order to alleviate this difficulty, major cloud providers (e.g., Amazon, Google, IBM, Microsoft) offer an interface to the prominent MapReduce [5] API for distributed computing to their users. MapReduce comprises not only parallelization ("Map") of work, but an aggregation ("Reduce") of individual results to keep computational burden on the user side low. While parallelization might be possible, it is unclear how related work on PIR can leverage aggregation.

This paper presents PIRMAP, an efficient PIR protocol suited for MapReduce clouds. PIRMAP especially targets retrieval of large files not considered in previous work. In a scenario with n files each of size l bits and $l \gg n$, PIRMAP achieves *optimal* communication complexity $O(l)$ with low constants. PIRMAP is designed for and leverages MapReduce parallelization and aggregation. We have implemented PIRMAP in Hadoop MapReduce, and its performance will be presented in Section 5.

Our contributions in this paper are:

1.) PIRMAP, a new, efficient PIR scheme for cloud computing with optimal $O(l)$ communication complexity when retrieving an l bit file for large l. PIRMAP runs on top of standard MapReduce, not requiring changes to the underlying cloud infrastructure.

2.) An implementation of PIRMAP that is usable in real-world MapReduce clouds today, e.g., Amazon. We evaluate PIRMAP, first, in our own (tiny) local cloud and, second, with Amazon's cloud. We verify its practicality with up to 1 TByte of data. Compared to the previously largest single database PIR experiment with up to 28 GByte of data [13], this demonstrates the efficiency and practicality of PIRMAP in the real-world. PIRMAP is more than one order of magnitude *cheaper* and *faster* than trivial PIR, and – in our particular scenario – we can significantly outperform related work. Compared to a theoretical, but unrealistic lower bound PIR, PIRMAP adds only 20% overhead on Amazon. PIRMAP's source code is available for download [15].

2. RELATED WORK

Initial PIR solutions were found with $O(l \cdot \sqrt{n})$ [8] communication complexity. Since then, much research has been done to reduce communication overhead down to poly-logarithmic complexity: see Cachin et al. [2], Lipmaa [9], or Ostrovsky and Skeith [14] for an overview. The lowest communication complexity reported today is $O(l \cdot \log n + k \cdot \log^2 n)$, where k is a security parameter [10]. However, this scheme has very large constants and results in a much larger communication cost for all practical values of n and l.

Despite the large amount of theoretical research, there has not been much investigation into practical PIR. Recently, Trostle and Parrish [20] compared the state of the art in and found that at best it took 8 minutes to retrieve a 3 MB file out of a dataset of 3 GB. Additionally, existing schemes have significant communication overhead for files of that size and larger.

3. PROBLEM STATEMENT

3.1 PIR

A PIR protocol is a series of interactions between the user and server that results in the user retrieving one file out of n, of his choice, while the server does not learn which file this was. More formally, the server cannot guess with probability greater than $1/n$ which file was queried. Note that this probability does not increase over multiple queries. This effectively hides the user's access pattern as each query is computationally indistinguishable from the others. The only information leaked is the number of records queried. In this short paper, we refrain from a more formal exposition of PIR and refer to Ostrovsky and Skeith [14].

3.2 MapReduce

With the trend towards more computers and more cores rather than faster individual processors, it is important that any practical PIR implementation be deployable in a way that can take full advantage of parallel and cluster computing environments. Perhaps the most widely adopted architecture for scaling parallel computation in public clouds today is Google's MapReduce [5]. Its design allows for a set of computations, or "job", to be deployed across many nodes in a cloud data center. Its biggest advantage is that it scales transparently to the programmer. That is, once an implementation is written using MapReduce, it will be able to run on any number of nodes in the data center, from one up to hundreds or thousands, without changes in the code. This is managed by splitting computation into two phases, each of which can be run in parallel on many computing nodes.

The first phase is called the "Map" phase. MapReduce will automatically split the input to the computation equally among available nodes in the cloud data center, and each node will then run a function called *map* on their respective pieces (called *InputSplits*). It is important to note that the splitting actually occurs when the data is uploaded into the cloud (in our case when the patient record/files are uploaded) and not when the job is run. This means that each "mapper" node will have local access to its InputSplit as soon as computation is started and you avoid a lengthy copying and distributing period. The map function runs a user defined computation on each InputSplit and outputs (emits) a number of *key-value* pairs that go into the next phase.

The second phase, "Reduce", takes as input all of the key-value pairs emitted by the mappers and sends them "reducer" nodes in the data center. Specifically, each reducer node receives a single key, along with the sequence of values emitted by the mappers which share that key. The reducers then take each set and combine it in some way, emitting a single value for each key.

Despite being widely used, MapReduce is a very specific computational model and not all algorithms can be easily adapted to it. A practical PIR such as PIRMAP has to take its specifics into account – as we will see below.

4. PIRMAP

PIRMAP is an extension of the PIR protocol by Kushilevitz and Ostrovsky [8], targeting the retrieval of large files in a parallelization-aggregation computation framework such as MapReduce. We will start by giving an overview of PIRMAP which can be used with any additively homomor-

Table 1: Cloud splits files into pieces

	1	2	$\cdots$	$\frac{l}{k}$
$file_1$	$B_{1,1}$	$B_{1,2}$	$\cdots$	$B_{1,\frac{l}{k}}$
$file_2$	$B_{2,1}$	$B_{2,2}$	$\cdots$	$B_{2,\frac{l}{k}}$
$\cdots$		$\cdots$		
$file_n$	$B_{n,1}$	$B_{n,2}$	$\cdots$	$B_{n,\frac{l}{k}}$

phic encryption scheme.

Upload. In the following, we assume that the cloud user has already uploaded its files into the cloud using the interface provided to them by the cloud provider.

Query. In keeping with standard PIR notation, our data set holds n files, each of which is l bits in length. There is also an additional parameter k which is the block size of the chosen cipher. For ease of presentation, we will consider the case where all files are the same length, but PIRMAP can easily be extended to accommodate variable length files by padding or prepending each file with a few bytes that specify its length. Our scheme is can be summarized as follows:

1. If the user wishes to retrieve file $1 \leq x \leq n$ out of the n files, it creates a vector $\vec{v} = (v_1, \ldots, v_n)$, where $v_x = \mathcal{E}(1)$ and $\forall i \neq x : v_i = \mathcal{E}(0)$. Here, $\mathcal{E}$ denotes any probabilistic, additively homomorphic encryption mechanism. The user sends $\vec{v}$ to the cloud.

2. The cloud arranges files into a table $\mathcal{T}$ similar to Table 1. The cloud divides each file i into $\frac{l}{k}$ blocks $\{B_{i,1}, \ldots, B_{i,\frac{l}{k}}\}$ and multiplies each block by v_i, i.e., $B'_{i,j} = v_i \cdot B_{i,j}$. Here, "$\cdot$" denotes scalar multiplication.

3. The cloud adds column-wise to create one result vector $\vec{r} = (r_1, \ldots, r_{\frac{l}{k}})$. Each element r_i of that vector is of size k, so the total bit length of $\vec{r}$ is l bits. Vector $\vec{r}$ is an encryption of file x. Vector $\vec{r}$ is returned to the user who decrypts it.

The cloud effectively performs the matrix-vector multiplication $\vec{r} = \mathcal{T}^T \cdot \vec{v}$, where $\mathcal{T}^T$ is $\mathcal{T}$ transposed. In practice the cloud does not need to create a table, but just needs to perform the block wise scalar multiplications and additions.

The computational complexity of this scheme is $O(n \cdot l)$, or the size of the whole data set. This is optimal because the cloud needs to "touch" every piece of data or else it would know certain files were not queried by the user. The communication complexity can be broken down into two parts: user to cloud and cloud to user. The user sends a vector of size n containing ciphertexts of size k, so the bandwidth complexity user-cloud is $O(n \cdot k)$. The cloud sends back a vector of $\frac{l}{k}$ entries, each of size $O(k)$, so cloud-user bandwidth complexity is $O(l)$. Consequentially, the overall communication complexity is $O(n \cdot k + l)$.

While PIRMAP appears to be relatively basic compared to other modern PIR schemes [2, 10], it achieve better overhead in practice because of the specific values of n and l that we will be using. Existing work in PIR is mostly concerned with datasets that have large values for n and comparatively small values for l. PIRMAP would do poorly in that setting, because the complexity would be dominated by n. However, PIRMAP's cloud-to-user communication is *optimal* at $O(l)$ because the cloud must send back a message at least the size of the file the user queries for. For values of l larger than n, PIRMAP allows for complexity of $O(l)$ with small constants. We argue that in practice this is often true. For example, if a user has a 1 TB dataset of 10 MB files, $\frac{l}{n} \approx 800$. In contrast, under the same conditions, arranging the files in a $\sqrt{n} \times \sqrt{n}$ matrix would result in a download cost of $1024 \cdot 10MB \approx 10GB$. Even if $n > l$, the actual communication costs are low for practical choices of the parameters, see Section 5.

Optimization: Although the cloud is doing most of the computation in this scheme, the user is still required to generate a vector of ciphertexts of length n and then decrypt the resulting response. Encryption is relatively expensive for ciphers that would be usable in this scheme, so it is a non-trivial amount of computation given that users could be low-powered devices such as smartphones. A way to alleviate this problem is to have a moderately powerful trusted server pre-generate vectors of ciphertexts and upload them to the cloud for the users to use. Optionally, it could instead be the low-powered device itself during an overnight downtime. This machine would generate m vectors of size n such that $V_{i,j} = \mathcal{E}(1)$ where $j = \text{HMAC}(k, i)$ and $V_{i,j} = \mathcal{E}(0)$ otherwise. This allows the user to use one of these "disposable" vectors at query time and permute it so that the single $\mathcal{E}(1)$ is at the index of the file it wishes to retrieve. If k is a key shared between the user and trusted server, the user can efficiently locate $\mathcal{E}(1)$. The user then generates a description of a permutation which moves the $\mathcal{E}(1)$ value to the correct position and randomly shuffles all other locations. A description of this permutation is of size $n \cdot \log(n)$, which is smaller than the size of the vector for $k > 30$, so this also effectively front loads the upload cost of the query and makes response time faster.

Although the encryption scheme we use (see Section 4.2 below) can perform encryptions very quickly and does not require the use of this optimization, we point it out as a general improvement that could be used with our scheme if the homomorphic cipher was changed.

4.1 PIRMAP Specification

In our protocol, the cloud performs two operations: multiplication of each block by the corresponding value in the "PIR vector" $\vec{v}$, and column-wise addition to construct the encrypted file chosen by the user. These two stages translate exactly to map and reduce implementations respectively. The files will be distributed evenly over all participating nodes where the map function will split each file into blocks and multiply the blocks by the correct encrypted value. The output of these mappers is a set of key-value pairs where the key is the index of the block and the value is the product of the block and encrypted PIR value. These values are all passed on to the reducers, which take a set of values for each key (block position or column) and add them together to get the final value for each block.

Being interested in $file_x$, the user executes algorithm GenQuery to compute $\vec{v}$ that it sends to the MapReduce cloud. There, each mapper node evaluates Map on its locally stored file and generates key-values pairs for the reducer. The reducer simply adds all values and sends them back to the user. The user receives $\frac{l}{k}$ values of size k from the reducers that he decrypts to get $file_x$.

```
User
function GenQuery($n, x$)
    $v := \{\}$
    for $i = 1$ to $n$ do
        if $i = x$ then
            $v_i := \mathcal{E}(1)$
        else
            $v_i := \mathcal{E}(0)$
        end if
    end for
end function
```

```
Cloud
function Map($file, v$)
    for $i = 1$ to $n$ do
        $c := B_i \cdot v_i$
        $Emit(i, c)$
    end for
end function
function Reduce($key, v$)
    $total := 0$
    for $i = 1$ to $n$ do
        $total := total + v_i$
    end for
    $Emit(key, total)$
end function
```

4.2 Encryption Scheme

Since the map phase of our protocol involves multiplying every piece of the dataset by an encrypted PIR value, it is important that we choose an efficient cryptosystem. Traditional additively homormorphic cryptosystems, such as Paillier's, have a form of multiplication as their homomorphism. That is, for some a and b, $\mathcal{E}(a) \cdot \mathcal{E}(b) = E(a + b)$. Since our map phase consists of multiplying ciphertexts by unencrypted scalars, we would actually have to do exponentiation of a ciphertext. Our scheme, and all PIR schemes, must compute on the whole dataset, so it would be quite computationally intensive to use a cipher requiring exponentiation.

Our solution to this problem is to use the *somewhat homomorphic* encryption scheme introduced by Trostle and Parrish [20] which relies on the hardness of the trapdoor group assumption. True homomorphic encryption schemes support an unlimited number of computations without increasing the size of the ciphertexts. In contrast, this scheme results in ciphertexts which grow in size by $O(\log_2 n)$ bits for n additions. In return for this size increase, we can have an encryption scheme where the additive homomorphism is actually addition itself. This scheme to encrypt n bits with some security parameter $k > n$ is as follows:

$KeyGen(1^k)$: Generate a prime m of k bits and a random $b < m$.
Encrypt $\mathcal{E}(x) = b \cdot (r \cdot 2^n + x) \mod m$, for a random r
Decrypt $\mathcal{D}(c) = b^{-1} \cdot c \mod m \mod 2^n$

This encryption has the desired homomorphic property $\mathcal{E}(a) + \mathcal{E}(b) = \mathcal{E}(a + b)$. This scheme is *somewhat homomorphic*, because it cannot support an unlimited number of additions. When two ciphertexts c_1 and c_2 are added, you can express the sum as

$$b \cdot (r_1 \cdot 2^n + x_1) + b \cdot (r_2 \cdot 2^n + x_2) = b \cdot ([r_1 + r_2] \cdot 2^n + x_1 + x_2).$$

If the inside term $(r_1 + r_2) \cdot 2^n + x_1 + x_2$ exceeds m and "wraps around", then it will not be decrypted correctly because application of the modulus will cause a loss of information. The modulus m must be chosen large enough to support the number of additions expected to occur. To support t additions, m should be increased by $\log_2 t$ bits. Additionally, each scalar multiplication can be thought of as up to 2^n additions, meaning that the size of m must be doubled for each supported scalar product. For our PIR scheme, m must be chosen to be $O(2k + \log_2(n))$ to support the required homomorphic operations.

In return for the reasonable increase in ciphertext size caused by the larger modulus (about 300% in our evaluations in Section 5), we are able to do very efficient computations over the encrypted data. Additionally, encryption is equivalent to only two multiplications, an addition and a modular reduction, while decryption is one multiplication and a reduction. This compares very favorably with other homomorphic encryption schemes, such as Paillier, requiring a modular exponentiation.

4.3 Privacy Analysis

PIRMAP inherits privacy properties of the work it is based on, i.e., traditional cPIR by Kushilevitz and Ostrovsky [8] and the PIR variant by Trostle and Parrish [20]. In the following, we sketch our privacy rationale.

PIRMAP is privacy-preserving, **iff** an adversary (the cloud) cannot guess, after each query, with probability greater than $1/n$, which file was retrieved by the user after an invocation of the protocol. There are two pieces of information that the adversary has access to: the set of uploaded files and the vector $\vec{v}$ of PIR values. If the files are each individually encrypted with an IND-CPA cipher, e.g., AES-CBC, they are computationally indistinguishable from random to the adversary. That means any attack that could break PIRMAP using the information in the database could also break it when run with a simulator that generates a set of random files. Therefore, privacy is dependent only on $\vec{v}$.

Vector $\vec{v}$ contains many encryptions of "0" and one encryption of "1". The problem of determining which file was selected is then equivalent to *distinguishing* between encryptions of "0" and encryptions of "1" in the underlying encryption. However, the scheme we use is provably secure against distinguishing under the Trapdoor Group Assumption [20]. Consequently, PIRMAP preserves user privacy.

5. EVALUATION

We have evaluated our scheme in two contexts: on a local "cloud" (a single server with multiple CPUs) and on Amazon's EC2 cloud using Elastic MapReduce [1]. PIRMAP has been implemented in Java for standard Hadoop MapReduce version 1.0.3 and is available for download [15].

5.1 Setup

Local: First, we have used a local server to prototype and debug our application and to do detailed timing analysis requiring many runs of MapReduce. This server, running Arch Linux 2011.08.19, has dual 2.4 GHz quad-core Xeon E-5620 processor and 48 GB of memory. Based on specs and benchmark results, our local server is closest to an "EC2 Quadruple Extra Large" instance, which has dual 2.9 GHz quad-core Xeon X-5570 processors and 24 GB of memory.

We have measured the time for PIR queries, i.e., the time to upload PIR vector $\vec{v}$ plus the time to process the query and download the result. Using Amazon's standard cost model, we have calculated the price of each PIR query as the amount of money required to run the query on one of the above EC2 instances [1] (for the same amount of time it took to run locally) plus the bandwidth cost of downloading the results [1]. Uploading data is free. To put our measurements into perspective, we have also included the time and cost of two other, hypothetical, PIR protocols. We have implemented a *Baseline*, which does not perform *any* cryptographic operations and merely "touches" each piece of data through the MapReduce API. This measure shows the theoretical *lower bound* of computation and time required for

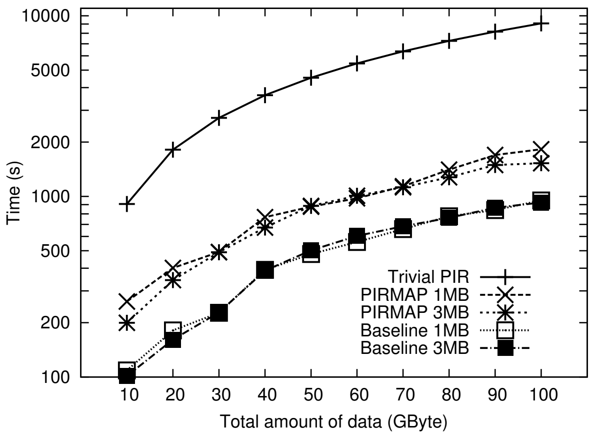

Figure 1: Time per query, local server

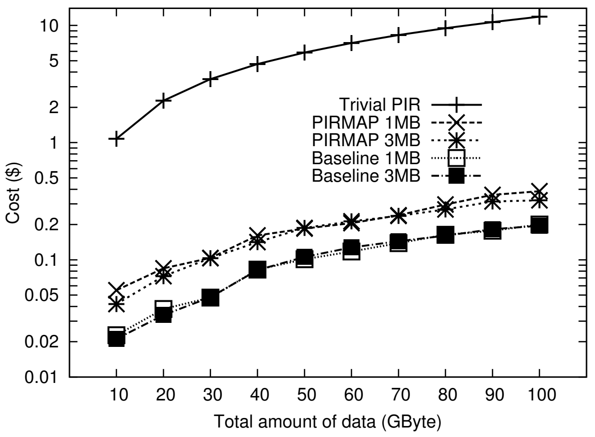

Figure 2: Cost per query, local server

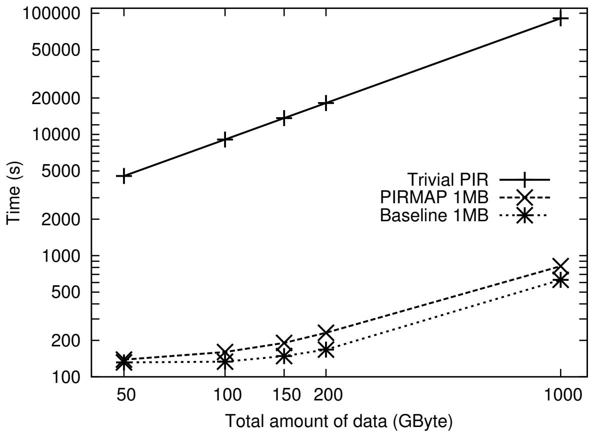

Figure 3: Time per query, Amazon Elastic MapReduce

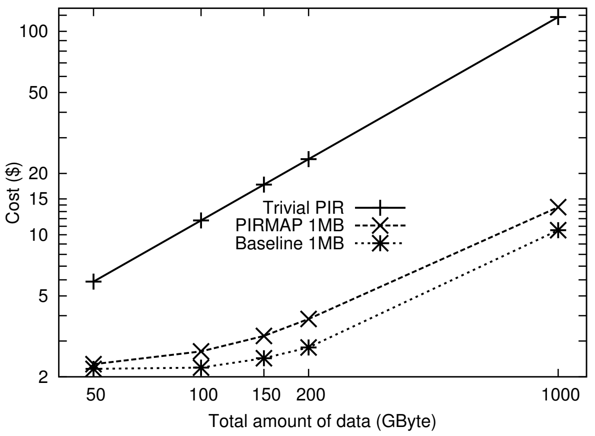

Figure 4: Cost per query, Amazon Elastic MapReduce

any PIR scheme that uses MapReduce, independent of the encryption and exact PIR method used.

To highlight the advantage of computational PIR, we have also included the time and cost required for the "trivial" PIR scheme. The trivial scheme is one where the user downloads the entire data set and simply discards the files he is not interested in. This is very bandwidth intensive, but computationally lightweight. Sion and Carbunar [16] conjecture this trivial PIR to be the most cost effective in the real-world. We have calculated the cost based on the amount that Amazon charges to download the corresponding amount of data and the time based on a 11.28 Mbps connection, an average as reported by Nasuni [12]. Note that we generally do not count the cost for long-term storage of data at Amazon. Although potentially significant for large amounts of data, the user has to pay for this regardless of whether he wants queries to be privacy-preserving or not. PIRMAP does not increase the amount of storage in Amazon.

Amazon: Besides the local experiments, to demonstrate the scalability of our scheme, we have also evaluated it on Amazon's Elastic MapReduce cloud. Amazon imposes a maximum limit of 20 instances per MapReduce job by default. In keeping with this restriction, we used 20 "Cluster Compute Eight Extra Large" instances which each having dual eight-core Xeon E5-2670 processors and 64 GB of RAM.

5.2 Results

Time and total cost: Figures 1 to 4 show our evaluation results. Figures 1 and 2 show the local evaluation, while figures 3 and 4 show evaluation with Amazon. In each figure, the x-axis shows the total amount of data stored at the cloud, i.e., number of files n times file size l. The y-axis shows either the time elapsed (for the whole query from the time the user submits the query until MapReduce returns the result back) or the cost implied with the query. In all four graphs, we scale the y-axis logarithmically, and in figures 3 and 4 we also scale the x-axis logarithmically. Each data point represents the average of at least 3 runs. Relative standard deviation was low at $\approx 5\%$.

To verify the impact of varying file sizes l, for our local evaluation, we show results with file sizes of 1 MB and 3 MB, attaining approximately equal runtime in both cases. This is to be expected because, in each data point, we have fixed the size of the database so varying retrieval sizes merely reshapes the matrix without changing the number of elements in it. The execution is dominated by the scalar multiplications that occur during the map phase, and the same number of those is required no matter the dimension of the matrix.

Our evaluation shows that PIRMAP outperforms trivial PIR in both time and cost by one order of magnitude. Compared to the theoretical, yet unrealistic optimum Baseline, PIRMAP introduces only 20% of overhead in the case of Amazon. We experience slightly larger overhead of 100%. This is because executing on Amazon has a much higher "administrative" cost due to the higher number of nodes and more distributed setting. These results indicate not only PIRMAP's efficiency over Trivial PIR and Baseline, but also its real-world practicality: in a small database comprising 10,000 patient record of size 1 MB each (10 GByte), a doctor can retrieve a patient record in ≈ 3 min for only $\approx \$0.03$. In a huge data set with $1,000,000$ files, a single file can be retrieved in ≈ 13 min for $\approx \$14$. In the case where it is necessary to retrieve data in a fully privacy-preserving manner, we conjecture this to be acceptable.

Although a comparison with related research is not straightforward (as PIRMAP targets a very special scenario), we put our results into perspective with those of Olumofin and Goldberg [13]. They use similar hardware (eight 2.50 GHz Intel Xeon E5420 CPUs), and can query a 30 GB dataset in ≈ 1000 sec. This about twice as slow as PIRMAP. It is also worth noting that their method takes advantage of GPU resources, whereas ours currently does not (we plan to address this in future research).

Query Generation and Decryption: Due to the efficiency of the encryption in PIRMAP, PIR query generation is very fast. One ciphertext (element of $\vec{v}$) is generated for each file in the cloud, so the generation time is directly proportional to the number of files. We omit in-depth analysis, but in our trials on a commodity Macbook running on a single roce it takes about 2.5 seconds per 100,000 files in the cloud. Decryption is slightly more expensive than encryption, but we still managed, on the same machine, to decrypt approximately 3 MB per second. We conclude this overhead to be feasible for the real-world.

Bandwith: PIRMAP introduces bandwith overhead, 1.) to *upload* $\vec{v}$, and 2.) to *download* the encrypted version of the file. For security, we set $k = 2048$ bit, so each of the n elements of vector $\vec{v}$ has size 2048 bit. For a data set with 10,000 files (10 GByte), this requires the user to upload ≈ 2.5 MByte. As this can become significant with larger number of files, we suggest to then use the optimization in Section 4, especially for constrained devices. With our choice of k, download of a 1 MB file increases to downloading a total of 3 MByte.

6. CONCLUSION

Retrieval of previously outsourced data in a privacy-preserving manner is an important requirement in the face of an untrusted cloud provider. PIRMAP is the first practical PIR mechanism suited to real-world cloud computing. In the case, where a cloud user wishes to privately retrieve large files from the untrusted cloud, PIRMAP is communication efficient. Designed for prominent MapReduce clouds, it leverages their parallelism and aggregation phases for maximum performance. Our analysis shows that PIRMAP is an order of magnitude more efficient than trivial PIR and introduces acceptable overhead over non-privacy-preserving data retrieval. Additionally, we have shown that our scheme can scale to cloud stores of up to 1 TB on Amazon's Elastic MapReduce service.

References

[1] Amazon. Elastic MapReduce, 2010. `http://aws.amazon.com/elasticmapreduce/`.

[2] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In *Advances in Cryptology, EUROCRYPT*, pages 402–414, Prague, Czech Republic, 1999.

[3] Y. Chen and R. Sion. On securing untrusted clouds with cryptography. In *Workshop on Privacy in the Electronic Society*, pages 109–114, Chicago, USA, 2010.

[4] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private Information Retrieval. In *Proceedings of Symposium on Foundations of Computer Science*, pages 41–51, Milwaukee, USA, 1995.

[5] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of Symposium on Operating System Design and Implementation*, pages 137–150, San Francisco, USA, 2004.

[6] Gartner. Gartner Identifies the Top 10 Strategic Technologies for 2011, 2010. `http://www.gartner.com/it/page.jsp?id=1454221`.

[7] Google. A new approach to China, 2010. `http://googleblog.blogspot.com/2010/01/new-approach-to-china.html`.

[8] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *Proceedings of Symposium on Foundations of Computer Science*, pages 364–373, Miami Beach, USA, 1997. ISBN 0-8186-8197-7.

[9] H. Lipmaa. An Oblivious Transfer Protocol with Log-Squared Communication. In *Conference on Information Security*, pages 314–328, Taipei, Taiwan, 2005.

[10] H. Lipmaa. First CPIR Protocol with Data-Dependent Computation. In *Conference on Information Security and Cryptology*, pages 193–210, Seoul, Korea, 2009.

[11] D. McClure. GSA's role in supporting development and deployment of cloud computing technology, 2010. `http://www.gsa.gov/portal/content/159101`.

[12] Nasuni. State of Cloud Storage Providers Industry Benchmark Report, 2011. `http://cache.nasuni.com/Resources/Nasuni_Cloud_Storage_Benchmark_Report.pdf`.

[13] F.G. Olumofin and I. Goldberg. Revisiting the Computational Practicality of Private Information Retrieval. In *Proceedings of Financial Cryptography*, pages 158–172, Gros Islet, St. Lucia, 2011.

[14] R. Ostrovsky and W.E. Skeith. A survey of single-database private information retrieval: techniques and applications. In *Proceedings of international conference on Practice and theory in public-key cryptography*, pages 393–411, Beijing, China, 2007.

[15] PIRMAP. Source Code, 2012. `http://www.ccs.neu.edu/home/travism/PIRMAP.zip`.

[16] R. Sion and B. Carbunar. On the Computational Practicality of Private Information Retrieval. In *Proceedings of Network and Distributed Systems Security Symposium*, pages 1–10, San Diego, USA, 2007.

[17] Symantec. State of Cloud Survey, 2011. `http://www.symantec.com`.

[18] Techcrunch. Google Confirms That It Fired Engineer For Breaking Internal Privacy Policies, 2010. `http://techcrunch.com/2010/09/14/google-engineer-spying-fired/`.

[19] The Telegraph. Patient records go online in data cloud, 2011. `http://www.telegraph.co.uk/`.

[20] J. Trostle and A. Parrish. Efficient computationally private information retrieval from anonymity or trapdoor groups. In *Proceedings of Conference on Information Security*, pages 114–128, Boca Raton, USA, 2010.

[21] Z. Whittaker. Microsoft admits Patriot Act can access EU-based cloud data, 2011. `http://www.zdnet.com/`.