Loco-Store: Locality-Based Oblivious Data Storage

Wenlong Tian, *Student Member, IEEE*, Ruixuan Li, *Member, IEEE*, Zhiyong Xu, *Senior Member, IEEE*, and Weijun Xiao, *Senior Member, IEEE*

Abstract—With the growing popularity of cloud storage, how to prevent information leakage from cloud access patterns attracts great attention. Oblivious RAM is proposed for this purpose. It is designed for the memory system, and most existing work focused on improving performance in the main memory. Recently, ORAM has been extended to the cloud environment, and it is called Oblivious Data Storage. TaoStore, the state-of-the-art oblivious data storage system, integrates the ORAM technology with synchronous I/O technology to reduce the mean response time. As we observed, there is a strong locality existing in user accesses. However, existing Oblivious Storage research did not consider this. In this article, we propose Loco-Store, an oblivious data storage. In Loco-Store, we design a novel stash controller scheme that can dynamically group relevant blocks during the oblivious I/O processes. We also propose a locality-based eviction algorithm to keep the security guarantee. The theoretical proof proves that our scheme keeps the security definition of ORAM. Finally, we implement a prototype and conduct extensive experiments on real-world datasets. The results show that Loco-Store can save the network bandwidth consumption up to 39.19 percent, and reduce the overall access time by 26.17 percent

Index Terms—Oblivious data storage, cloud storage, spatial locality, temporal locality

1 Introduction

With the rapid growth of cloud storage, the demand for security and privacy of user's outsourced data keeps increasing. Although the data content can be protected by encryption schemes, recent research paper [1] shows that user's access patterns in cloud scenarios can still result in private data leakage to the cloud server.

Traditionally, to prevent the leakage of user's privacy information, Oblivious RAM (ORAM) was proposed in 1987 to make memory access pattern oblivious [2]. But, it is not widely used in real world because of its unacceptable performance. Recently, Path ORAM [3] has attracted attention in the research community. It has a simplified algorithm and outperforms previous ORAM solutions significantly. However, it still incurs at least 30$\times$ more latency than normal memory access. Wang *et al.* [4] takes advantage of the buffer-on-board (BOB) like memory architecture to offload the ORAM operations into a secure engine. Shafiee *et al.* [5] reduce the ORAM access latency by designing a secure DIMM. Zhang *et al.* [6] and Fujieda *et al.* [7] reduce the number of duplicated requests by removing unnecessary oblivious accesses. There are many other related works. Most of them are focused on the cache level in a single computer with small and fast memory.

In the cloud environment, data are shared by multiple users. Thus, the security concern on data access pattern leakage is more severe. Therefore, ORAM technology has been extended to relieve this issue in cloud storage. It is denoted as Oblivious Data Storage. However, directly applying the ORAM technology will induce inferior I/O performance. Correctly, to keep the obliviousness property of ORAM technology, each block request is transmitted into a fetching operation including itself and a series of other irrelevant blocks. It results in lots of extra I/O accesses. Here we defined that relevant blocks have strong spatial locality, and hot blocks have high temporal locality. To improve the performance, Stefanov *et al.* proposed a distributed ORAM-based Cloud Store by making I/O operations asynchronous [8]. Sahin *et al.* made a further improvement and designed a tree-based asynchronous oblivious store by considering asynchronous network communication and concurrent processing of requests [9]. Although asynchronous I/O can improve the throughput by decreasing the response time, the obliviousness property makes it difficult to hold hot blocks and waste lots of network transmission for irrelevant blocks.

As we know, the data access operations on file or block-level show high temporal and spatial locality in real-world scenarios [10]. In the cloud environment, users' data access shows strong data locality. If we can store previously accessed blocks or relevant blocks, and they are requested soon, we do not have to retrieve from the server again and significantly reduce network bandwidth consumption. However, existing oblivious data storage designs do not consider locality.

- *Wenlong Tian is with the School of Computer Science and Technology, the University of South China, Hengyang, Hunan 421001, China. E-mail: wenlongtian@usc.edu.cn.*
- *Ruixuan Li is with the School of Computer Science and Technology, the Huazhong University of Science and Technology, Wuhan, Hubei 430074, China. E-mail: rxli@hust.edu.cn.*
- *Zhiyong Xu is with the Math and Computer Science Department, Suffolk University, Boston, MA 02108 USA , and also with the Shenzhen Institute of Advanced Technology, the Chinese Academy of Science, Beijing 518000, China. E-mail: zxu@suffolk.edu.*
- *Weijun Xiao is with the Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, VA 23284 USA. E-mail: wxiao@vcu.edu.*

Manuscript received 19 Oct. 2019; revised 2 July 2020; accepted 9 July 2020. Date of publication 15 July 2020; date of current version 14 Mar. 2022. (Corresponding author: Ruixuan Li.) Digital Object Identifier no. 10.1109/TDSC.2020.3009428

1545-5971 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

ORAM randomly groups the blocks based on the oblivious property to hide the access patterns. Thus, each oblivious access contains multiple irrelevant blocks. Furthermore, the random eviction process in ORAM does not align successive blocks together as well. The stash in ORAM can hardly hold relevant blocks together since the sets of read/write blocks are always selected randomly.

In this paper, we are inspired by the successful adoption of utilizing locality in storage domains, and attempt to integrate locality and obliviousness, these two technologies together. We propose a new oblivious data storage architecture, called Loco-Store. In Loco-Store, instead of hiding the access pattern by randomly grouping the blocks, we select relevant blocks and put them together to improve the performance. Meanwhile, our design also keeps the obliviousness feature in the original ORAM design. Thus, we can keep the security promise. More specifically, we renovate the stash controller design to reduce the number of oblivious accesses by dynamically grouping sequentially accessed blocks together to take advantage of spatial locality. It also holds hot data blocks together to utilize the temporal locality. Besides, a locality-sensitive eviction scheme is also proposed. The main contributions of this paper are summarized as follow:

- First, we analyze ORAM technology and present its limitations. Traditional ORAM technology can hardly align successive blocks together. It cannot retain hot blocks for future accesses as well. Thus, existing oblivious data storage build on top of ORAM cannot utilize the locality property. Each oblivious access only contains irrelevant blocks.
- Second, we design a new oblivious data storage architecture, Loco-Store, to integrate the features of locality and obliviousness in the multi-user environment. In Loco-Store, a novel stash controller taking advantage of both temporal and spatial locality is designed, and a locality sensitive eviction scheme is also proposed. Meanwhile, Loco-Store can still maintain provable security as traditional Oblivious Data Storage.
- Third, we theoretically formalize our design and justify its security feature. We prove that, in the cloud environment, the access pattern in Loco-Store is computationally indistinguishable with a random sequence of bit strings.
- Finally, we conduct extensive simulations to verify the effectiveness of Loco-Store. The experimental results show that Loco-Store outperforms the state-of-the-art Oblivious Data Storage solution, TaoStore. It can save the network bandwidth up to 39.19 percent, and reduce the overall access time by 26.17 percent.

The remaining sections are structured as follows. In Section 2, we discuss the related work. Then, we introduce the threat model, Path ORAM, and the limitations of ORAM technology in Section 3. Next, we design a novel locality-based Oblivious Data Storage, Loco-Store, to integrate the locality and obliviousness in Section 4. The proof of Loco-Store security property is presented in Section 5, and the experimental results are analyzed in Section 6. Finally, we conclude the paper in Section 7.

2 Related Work

In this section, we mainly discuss the related work about Oblivious RAM. We categorize these related work into three categories, such as the ORAM algorithm's optimization, ORAM with novel architectures, and the oblivious Data storage by utilizing ORAM technology.

ORAM Algorithm's Optimization. Although the first Oblivious RAM was proposed by Goldreich and Ostrovsky for address obfuscation in 1987 [2], Path ORAM [3] is a milestone to saving the bandwidth by its simple and effective algorithm. It formally proved that the algorithm's security. Since then, numerous works have attempts to improve the performance of the ORAM. For example, Ring ORAM [11], Bucket ORAM [12] were proposed to reduce the bandwidth overhead on the memory bus by using different bucket organization and more complicated access flow control. To further improve Path ORAM performance, several techniques have been proposed. Ren *et al.*[13] optimized block mapping using sub-tree layout, which maximizes row buffer hit for ORAM accesses. They saved the top of the Path ORAM tree in a small on-chip cache to improve performance. Zhang *et al.* [6] eliminated unnecessary memory accesses if consecutive path accesses have overlaps. Wang *et al.* [14] proposed an efficient bandwidth sharing technique, and read and write phase acceleration for ORAM applications co-run with other applications on a server with the conventional memory interface. Ren *et al.* [15] has improved the performance of ORAM by introducing the static super block structure and the dynamical adaptive algorithm. It saves the bandwidth in ORAM based on spatial locality property.

ORAM With Novel Architectures. Some other alternatives (e.g., [16], [17], [18], [19]) leveraged the cooperative computation with servers or other hardware to further reduce the communication cost. For example, Onion-ORAM [16] decrease the bandwidth blowup, where the client and server interactively run partial homomorphic encryption operations. Mayberry *et al.* [17] used PIR scheme in [20] with additively homomorphic encryption (AHE) (i.e., [21]) on top of tree ORAM structure [22]. The scheme in [23] used PIR scheme in [20] on top of ObliviStore [8], which is based on Partition-ORAM in [24]. The S^3ORAM in [25] proposed a distributed ORAM based on multi-server architecture by introducing the Shamir Secret Sharing scheme. Besides, Ali *et al.* [5] try to improve the ORAM performance by introducing secure dual in-line memory module (SDIMM), which use a 64-bit data path. It can leverage the SDIMM to reduce bandwidth, latency, and energy per ORAM access. Meanwhile, Rujia *et al.* [4] also improve the ORAM performance by taking a similar idea. They utilize the buffer-on-board (BOB) memory architecture as the secure delegator to achieving high privacy protection, and speedup ORAM access. Chandrasekhar *et al.* [26] uses a two-level ORAM and reduces the overhead for the first level by packing most metadata into space typically used for Error-Correction Codes. Cao *et al.* [27] propose a blockchain-based traceable ORAM to detect malicious behavior. It utilizes the untampered property of blockchain to integrate the group signature with oblivious access.

Oblivious Data Storage by Utilizing ORAM Technology. In the cloud storage scenario, the semi-honest cloud server is also eager to master the user's access pattern. But, most of

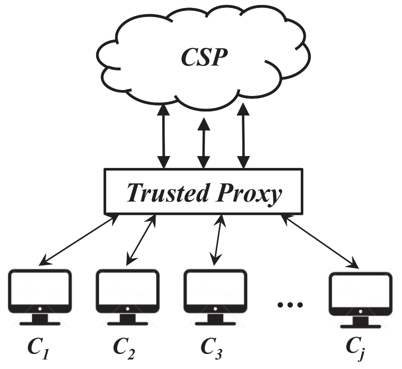


Fig. 1. Deployment model of loco-store.

the existing previous ORAM works are improve the ORAM performance in the CPU cache level, which always focused on single-client, sequential ORAM and do not adaptive the I/O characters in the cloud storage scenario. Stefanov *et al.* propose a distributed ORAM-based Cloud Store by making I/O operations asynchronous [8]. Sahin *et al.* also propose a tree-based asynchronous oblivious store by considering asynchronous network communication and concurrent processing of requests [9]. Although asynchronous mode can greatly decrease the response time, it is difficult to reduce the soaring network bandwidth overhead caused by ORAM technology. Moreover, substantial files or blocks' accesses show strong temporal locality and strong spatial locality in real scenarios while the existing oblivious storage design ignores the positive effect from the locality property, which is the focus of our work.

3 Background

In this section, we describe the threat model used in this paper. Then, we elaborate the access pattern protection process in Path ORAM, as a representative of traditional ORAM technology [3]. Finally, we discuss the limitations of traditional ORAM solutions.

3.1 Threat Model

In Loco-Store, the threat model is based on a trusted proxy architecture. As shown in Fig. 1, the trusted proxy is acted as a middle layer between clients and the cloud service provider (CSP). The proxy coordinates all user accesses into the oblivious accesses. The data is encrypted before uploading it to the cloud storage. We assume that the cloud storage provider is honest-but-curious, and the trusted proxy server has a reasonably large memory. The communications between the trusted proxy and users are protected by end-to-end encryption. In real life, the proxy model is widely used. For example, the company and university can match this threat model. All the staffs and students connect to the Internet through the proxy (organization gateway). This threat model is widely used in many research works. And we take the same threat model as well [8], [9].

Under this threat model, there are two potential vulnerabilities. The first one is that an attacker can compromise with the cloud service provider. Second, an attacker can also initiate the man-in-the-middle attack and monitor the cloud server for client access patterns. It is noted that there is another threat that information leakage through timing channel [28]. Like other research work, we only consider the first two attacks. We will leave the third one as our future work.

3.2 Path ORAM

In ORAM, to prevent the private information leakage from access patterns, each access should be transformed into oblivious access. Thus, from the view of attackers or the cloud server, the oblivious access is computationally indistinguishable with a random sequence of bit strings. Path ORAM is one of the most popular ORAM implementations.

In Path ORAM [3], a complete binary tree maintains the data blocks with the root at level 0 and leaves at level L. Each node in the tree is a bucket of size Z. There are at most Z real data blocks in each bucket, while all the others are dummy blocks. Each block is encrypted by probabilistic encryption [29]. Each leaf node has a leaf ID, and its values vary from 0 to 2^{L-1}. And, there is a position map in Path ORAM. It is a lookup table that records the association between each data block and a leaf ID if this data block is on the path. The blocks in the intermediate node are on multiple paths. The system randomly chooses one leaf ID from the paths.

For each oblivious access, the client locates the target block's leaf ID. Then, according to the ID, all the blocks on the path are fetched. Only the real data blocks are decrypted and stored in the stash. The stash is a small memory buffer on the client side. Meanwhile, the client will randomly assign a new leaf ID for the target block and update the local position map. At the end of each oblivious access, several blocks in the stash will be selected and written back to the original path. The blocks to be selected must have an intersection with the path of the original leaf ID. Finally, these blocks are re-encrypted and written back to the original path. The details can be found in [3].

3.3 The Limitations of ORAM Technology

To prevent the user's privacy leakage from access patterns, each access in ORAM includes a read and a write operation. Requesting a block is transformed into the operation of fetching multiple blocks. Thus, the adversary cannot figure out which block is the target block and whether the user's operation is a read or a write. For example, each oblivious access in the Path ORAM must fetch and put 60 blocks when the block number in a bucket equals four, and the level of ORAM tree is 15. As shown in Fig. 2, blocks 1 to 6 are randomly organized in an ORAM tree. We use the same color for relevant blocks. When a client initiates a series of successive block requests from block 1 to 6, Path ORAM induces six oblivious accesses. Furthermore, in each oblivious access, the fetched blocks are randomly written back to the ORAM tree. The stash is usually empty after access, as proved in [3]. Path ORAM does not consider temporal and spatial locality. Relevant blocks can hardly be grouped together.

When applying the ORAM technology to cloud storage, the situation becomes even worse. In the multi-user scenario, a client does not know requests from other clients, and the server has little knowledge about the client's access pattern as well. The outsourced data is logically organized as a binary tree or forest. Thus, a single client or the server can hardly detect the frequently accessed blocks across

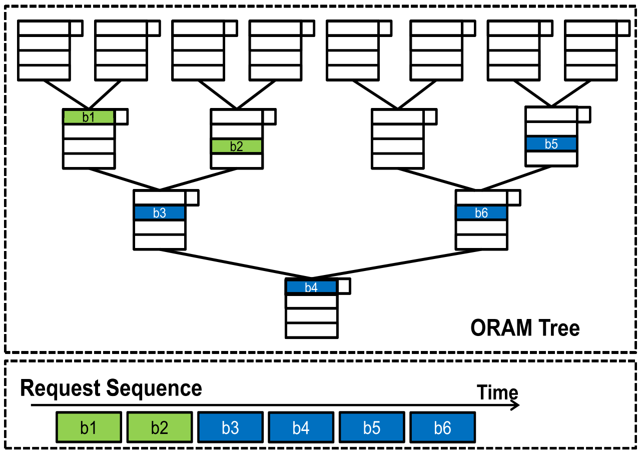


Fig. 2. Blocks 1 to 6 are randomly stored in ORAM tree and a request sequence accesses these blocks in order.

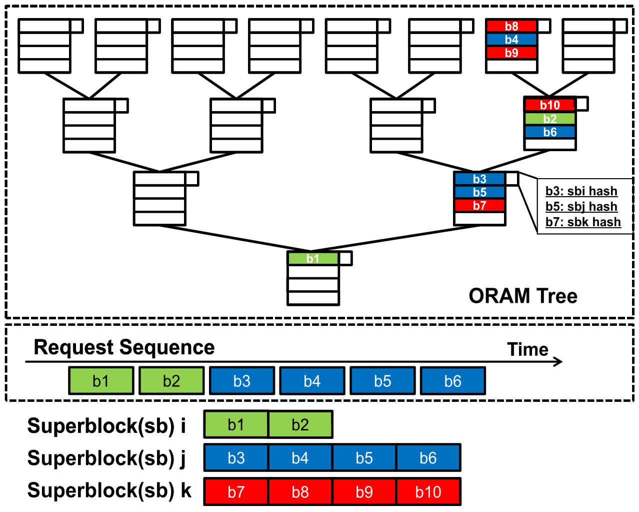


Fig. 3. The superblock structure.

multiple users. The system cannot utilize existing locality among users. To combining the locality with obliviousness, Yu *et al.*[15] is the first to propose a dynamic prefetcher for ORAM by introducing a superblock design. The blocks exhibit spatial locality are merged into a superblock. It is a big step in exploring how to utilize locality to improve ORAM performance. However, It still exists some disadvantages. First, It is designed at the cache-level for a single user scenario. Only the data blocks adjacent in a program address space are grouped as a superblock. Thus, it can hardly be applied in the Oblivious Data Storage. Second, It ignores the effect of the temporal locality on data accesses, especially in a multi-user scenario.

As the state-of-the-art Oblivious Data Storage, TaoStore [9] introduces asynchronous technology to reduce access latency. It allows multiple clients to securely and obliviously access their shared data on an untrusted storage server. Moreover, it guarantees both the contents of the shared data and the accesses from multiple users are kept hidden against any middle attacker observing traffic to and from the server. However, it barely retains relevant blocks and wastes a lot of network bandwidth for unrelated block transmissions. In the worst case, all relevant blocks are distributed into different paths. In this case, the number of oblivious accesses almost equals the number of data access requests received by the server. Undoubtedly, in Oblivious Data Storage, lots of block transmissions are unnecessary. It seriously downgrades the system performance.

4 Design of Loco-Store

In this section, we first describe the overview design of Loco-Store. Then, we present a novel stash controller. Finally, a locality-based eviction scheme is elaborated.

4.1 Overview

The main goal of the Loco-Store is to avoid unnecessary oblivious access by integrating the features of the locality and obliviousness in a multi-user environment. Thus, each data request in Loco-Store is converted by the trusted proxy into oblivious access. And the cloud service provider responds to the proxy with a target block and some relevant blocks. Then, the target block is returned to the client. Moreover, Loco-Store utilizes Loco-ORAM, a locality-based ORAM scheme to integrate the locality and obliviousness by introducing three modules: the interactive module, the stash controller, and the locality-based eviction module.

The interactive module has two functions. The first one is to fetch and decrypt all blocks of a path. Then, these blocks are put into the stash on the trusted proxy. The second one is responsible for encrypting data blocks. Then, these blocks are written back to the original path on the cloud server. The stash controller module is accountable to group multiple relevant blocks as a superblock based on temporal and spatial locality. Then, the locality-based eviction module randomly selects blocks to keep the obliviousness feature. Unlike traditional ORAM techniques, the selection in the locality-based eviction module is based on the superblock level. It is noted that Blocks within a superblock do not have to be stored in the same bucket, as shown in Fig. 3. Specifically, the subsequent request from block 1 to 6 in Loco-Store only needs one oblivious access. Moreover, Each bucket has extra space to store the metadata information, including the superblock hash values. Here, we keep the superblock hash value for each block in the metadata of this bucket. For each superblock, we calculate its hash value based on its data block contents.

Next, Algorithm 1 detailedly shows the oblivious access function in Loco-ORAM, which integrates the locality and obliviousness features by scheduling the above three modules. We denote it as the "Access" function. There are three parameters in this function such as 'op', 'a', and 'data*'. 'op' denotes the operation which is read or write. 'a' denotes the logical address of the target block. 'data*' denotes the content of the writing block. To simplify the description, the symbols used in Loco-ORAM are listed in Table 1. The oblivious access function is briefly summarized in steps:

1) *Hit block* (Lines 1 to 7): For each oblivious access, Loco-ORAM first detects whether the requested block locates in the stash S or not. Once there is a hit, it can dynamically group and manage the blocks of the stash by calling stash_Control function. The details of stash controller is in next Section 4.2. Then, the target block can be returned to the client.

TABLE 1
Notations

Notation	Description
N	Total blocks outsourced to server
L	Height of Loco-ORAM binary tree
Z	The number of Maximum blocks in each Bucket
$P(x)$	Path from leaf ID x to the root
$P(x,l)$	The bucket at level l along the path $P(x)$
S	The stash in Loco-ORAM(located at the trusted proxy)
$position$	Loco-ORAM's position map
$dummy\,block$	Randomly generated block
$remove\_size$	A threshold in locality-based eviction module

2) *Read path* (Lines 9 to 11): Once the target block is not hit in the stash S, Loco-ORAM can achieve its leaf ID from $position$. If the ID equals to null, Loco-ORAM will randomly generate a leaf ID. Meanwhile, all real blocks in a path from leaf ID to the root are fetched into the Stash S by the readPath function of the interactive module. Since the trusted proxy has a reasonable memory in the threat model, it stores the position map locally.
3) *Update block* (Lines 12 to 15): When the operation equals to a 'write', the target block in the stash is updated based on the new content data*. Then, whether "op" is a read or a write operation, the stash controller module is triggered by calling the Stash_Control function. It screens out the blocks that do not reference again in the near future, and dynamically group relevant blocks together.
4) *Write path* (Lines 16 to 20): To write back the rarely accessed blocks and keep the provable security, the locality-based eviction module is triggered by calling the locality_Eviction function. It is responsible for deciding how many blocks could be written back and construct a written back path. If the real block number of the path does not reach the maximum block number of the path, it will be padded by dummy blocks. Then, those evicted blocks are removed from S. The constructed path will be encrypted by the probabilistic encryption [29] and written back by the writePath function of the interactive module.

Subroutines. The readPath function is used to fetch the blocks from the cloud to the trusted proxy. Meanwhile, the writePath function writes back the blocks to the cloud. Both readPath and writePath functions contain the encryption and decryption process. Here, these cryptography schemes in Loco-ORAM are the probabilistic encryption [29] like Path ORAM.

4.2 Stash Controller

The stash controller introduces three queues, temporal queue, spatial queue, and removed queue to dynamically group the relevant blocks in the stash based on temporal and spatial properties. The overview of the stash controller is shown in Fig. 4. For each oblivious access, the target block is pushed into the temporal queue. The temporal queue maintains frequently accessed blocks by dynamically block replacement. Only the block, which may be re-accessed in the future, is held in the temporal queue. Here, The replacement algorithm can be realized by any cache replacement algorithm, such as Least recently used policy (LRU) [30] or Least Frequently Used policy (LFU) [31]. Besides, the spatial locality is maintained by the spatial queue. To further optimize the locality management, we add a removed queue to hold more blocks that might be accessed in the future.

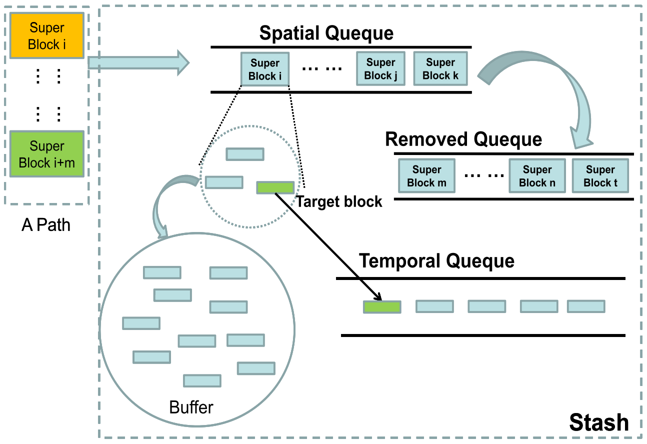


Fig. 4. The overview of the stash controller.

Algorithm 1. Access(op,a,data*)

```
1:  data← Read block a from $S$
2:  if data $\neq$ null then
3:    if op=write then
4:      data← data$^*$
5:    end if
6:    stash_Control()
7:    returndata
8:  else
9:    x← $position$[a]$\neq$null ? $position$[a] : random leaf ID
10:   $S \leftarrow$ readPath($P(x)$)
11:   data← read target block a from stash
12:   if op=write then
13:     $S \leftarrow$ ($S$-{(a,data)})∪ {(a,data*)}
14:   end if
15:   stash_Control()
16:   P(x)←locality_Eviction()
17:   $S' \leftarrow$ Select blocks from $P(x)$
18:   $S \leftarrow S - S'$
19:   writePath($P(x)$)
20: end if
```

Specifically, both the spatial and the removed queue are based on superblock to manage the input and output of queues. Each block in the temporal queue has a corresponding superblock. All the superblocks are kept in the spatial queue. Moreover, the rarely accessed superblock is removed from the spatial queue to the removed queue. For example, once a target block is accessed, the target block is pushed into the temporal queue. Meanwhile, the corresponding superblock is pushed into the spatial queue if it is not in the queue before. It is noted that whole blocks in the stash are stored at the buffer. Furthermore, each queue only maintains the reference of the block or superblock. The superblocks of these queues are dynamically merged and split based on the accessed frequency and relevance. Concretely, each superblock reference in the spatial queue records time to use (*TTU*) and its blocks' hit counts. If a new superblock is pushed into the spatial queue, all superblock records in the spatial queue are scanned for merging and splitting. Only the superblock, which *TTU* is larger than a predefined value, δ, can be chosen as a candidate for merging or splitting. The split superblock from the spatial queue will be pushed into the removed queue. Once a block in the superblock is re-accessed, it can re-push into the spatial queue. Next, we detailed introduce the merge and split process in the spatial queue.

To support the merge process, the stash controller monitors the request sequences in the trusted proxy. If there exist cross-accesses between two different superblocks, it records these events for a period time δ. Thus, according to these records, the stash controller merges the relevant two superblocks into a new superblock. For example, during a period of time δ, once a block i in a superblock k is accessed after the access of block j in other superblock m, the superblock j and m can be merged as a new superblock. It is noted that we defined that the size of each superblock does not exceed the $(L * Z)/4$, which is based on our experience. If the size of the new superblock after the merging process is larger than the maximum superblock size, the stash controller can directly terminate the merge operation.

$$CV_i = \frac{\sigma_i}{\mu_i}. \tag{1}$$

In the split process, the stash controller calculates the coefficient of variance (CV) for superblocks larger than δ in *TTU*. In probability theory and statistics, the coefficient of variance is a standardized measure of a frequency distribution dispersion. Here we use the coefficient of variance to select which superblock could be split. In other words, the high value of CV shows that some blocks in a superblock are accessed frequency and vice versa. Specifically, We denote that the block number in a superblock is N. The access probability for each block in this superblock is $\frac{1}{N}$. We can calculate the average access number and variance of blocks in the superblock i based on the stash controller's monitor, denoted as μ_i and σ_i. It is noted that the average access number means the average number of block accesses in a superblock. Thus, the CV_i of the superblock i can be achieved based on the Formula 1. Meanwhile, the stash controller calculates the average access number (AHN) of superblocks. Then, it filters out the superblocks with CV larger than a predefined threshold T and denoted these superblocks as a set G. If the μ_i of the superblock i in the set G is less than the three-quarter AHN, it is split into two superblocks. Blocks in the superblock i with the accessed number not less than μ_i are grouped as a new superblock. Finally, the superblock i is pushed into the removed queue from the spatial queue. Furthermore, once a block hit in a superblock of the removed queue, it gets put back on the spatial queue from the removed queue.

4.3 Locality-Based Eviction

The locality-based eviction module mainly consists of two subtasks to securely group relevant blocks into a written-back path. The first one is the leaf ID assignment. The second one is the relevant block selection for eviction. The former is the key to keeping the access pattern in Loco-Store computationally indistinguishable with a random sequence of bit strings in the cloud or attackers' view. The latter one is to write the relevant blocks on the same path. The detailed pseudocode of $Locality\_Eviction$ function is shown as Algorithm 2.

First, the $Locality\_Eviction$ function generates an empty path, $P(x)$, by padding $dummy\,blocks$ (line 1). If the length of the removed queue is less than a predefined threshold, $remove\_size$, the $Locality\_Eviction$ function directly returns $P(x)$(lines 2-3). If not, the function calculates how many real blocks could be written back by randomly generated leaf ID (from lines 5-19). When the leaf ID of $P(x)$ belongs to $[0, 2^{L-2})$, the generated leaf ID scope is $[0, 2^{L-2})$. When leaf ID of $P(x)$ belongs to $[2^{L-2}, 2^{L-1})$, the generated leaf ID scope is $[2^{L-2}, 2^{L-1})$. Then, if there are intersections between $P(x)$ and the path corresponding to randomly generated leaf ID, the intersection nearest to the leaf node could be recorded as the location where the data block can be written. Finally, the $Locality\_Eviction$ function can select K superblocks from the removed queue and write them to previous recording locations of $P(x)$ in sequence (from lines 20-26).

Algorithm 2. Locality_Eviction()

```
1: construct a path $P(x)$ by padding dummy blocks
2: **if** the length of Removed Queue is less than $remove\_size$ **then**
3:   return the path $P(x)$
4: **else**
5:   Count← the Removed Queue length $-\,remove\_size$
6:   **if** $0 <= x < 2^{L-2}$ **then**
7:     randomly generate $L * Z$ leaf ID in $[0, 2^{L-2})$ and store at the array R
8:   **else**
9:     randomly generate $L * Z$ leaf ID in $[2^{L-2}, 2^{L-1})$ and store at the array R
10:  **end if**
11:  evict_count←0
12:  **for** $l \in \{L, L$-1,$\cdots$,0$\}$ **do**
13:    **for** i∈0,1,2,$\cdots$, $Z$-1 **do**
14:      **if** $P(x, l)$==$P$(R[$l * Z$+i],$l$) **then**
15:        evict_count++
16:        record the writing location of the path $P(x)$
17:      **end if**
18:    **end for**
19:  **end for**
20:  **if** the block number of top $K$ superblocks in Removed Queue $<$ evict_count **then**
21:    pop $K$ superblocks from Removed Queue where $\sum_{i=1}^{K} superblock\_size_i <$ evict_count
22:    write blocks of $K$ superblocks sequentially into the recorded writing location of $P(x)$
23:    update the $position$ map
24:  **end if**
25:  return the path $P(x)$
26: **end if**
```

Note that Loco-ORAM is easily amenable to partitioning that ORAM tree in a distributed fashion across multi partitions as in the previous [9] and [3] . The only difference is the data-fetch logic. Moreover, all the blocks will be re-encrypted based on probabilistic encryption by the interactive module. The cloud server can not know information about: 1) which data is being accessed; 2) how old it is; 3) whether the same data is being accessed; 4) access pattern (sequential, random, etc); 5) whether the access is a read or a write. 6) what is the relevance among the blocks of a path. The detailed security proof of the Loco-ORAM is detailed in Section 5.

5 Security Proof

In this section, we detailedly prove that Loco-ORAM keeps the security definition of ORAM technology. We first take Path ORAM as an essential representative ORAM to give out a security definition of ORAM technology. Then, to simplify the proven process, we formulate the security definition in Path ORAM and Loco-ORAM. Finally, we discuss the

For any access in Path ORAM, it satisfies the following definition. Let $\vec{y} := ((op_M, a_M, data_M), \ldots, (op_1, a_1, data_1))$ denotes a data request sequence of length M, where each op$_i$ denotes a read(a$_i$) or a write(a$_i$,data) operation. Specifically, a$_i$ denotes the identifier of the block being read or written, and data$_i$ denotes the data being written. In our notation, index 1 corresponds to the most recent load/store, and index M corresponds to the oldest load/store operation. Let A($\vec{y}$) denotes the access sequence from the cloud's or attacker's perspective.

Definition 1 (Security Definition). *An ORAM construction is said to be secure if (1) for any two data request sequences $\vec{y}$ and $\vec{z}$ of the same length, their access patterns A(y) and A(z) are computationally indistinguishable by anyone but the client, and (2) the ORAM construction is correct in that it returns on input $\vec{y}$ data that is consistent with $\vec{y}$ with probability $\geq 1 - negl(\|\vec{y}\|)$.*

In other words, the cloud cannot acquire the following information based on the client's access pattern: a) which data is being accessed;b) how old it is (when it was last accessed); c) whether the same data is being accessed(linkability); d) user's truly access pattern (sequential, random, etc.); e) whether the access is a read or a write operation. Then, we formalize the distributed probability of A($\vec{y}$) in Path ORAM and Loco-ORAM. In Path ORAM, the scope of each block's new leaf ID in stash is $(0, 2^{L-1})$. Moreover, each evicted block is independent of others. Thus, we can get the formalization by using Bayes rule as follow:

$$Prob(A(\vec{y})) = \prod_{i=1}^{M} Prob(position(a_i)) = (2^{L-1})^{-M}. \tag{2}$$

Then, we use the similar idea to formalize $Prob'(A(\vec{y}))$ in Loco-ORAM. Based on the locality-based eviction scheme in Loco-ORAM, each eviction block's leaf ID is located at $[0, 2^{L-2})$ or $(2^{L-2}, 2^{L-1}]$. Thus, $Prob'(position(a_i))$ is equals to $(2^{L-2})^{-1}$ by using Bayes rule. Then, we can also get the probability of $A(\vec{y})$ in Loco-Store as Formula 3 since each leaf ID's generation of a_i is independence. We denote the probability of $A(\vec{y})$ in Loco-Store as $Prob'(A(\vec{y}))$.

$$Prob'(A(\vec{y})) = \prod_{i=1}^{M} Prob'(position(a_i)) = (2^{L-2})^{-M}. \tag{3}$$

To further prove that the access patterns in Loco-ORAM and Path ORAM are computational distinguishable, we give out the definition of computational distinguishability. In computational complexity, let X_n, Y_n be sequences of distributions with X_n, Y_n ranging from $0, 1^{l(n)}$ for some $l(n) = n^{O(1)}$. X_n and Y_n are computationally indistinguishable if for every non-uniform probabilistic polynomial-time algorithm A and following quantity is a negligible function in n:

$$\delta(n) = \left| \underset{x \leftarrow D_n}{Pr}[A(x) = 1] - \underset{x \leftarrow E_n}{Pr}[A(x) = 1] \right|. \tag{4}$$

It is noted that the negligible function is a function δ: $\mathbb{N} \leftarrow \mathbb{R}$ such that for every positive integer c there exists an integer N_c such that $|\delta(x)| < \frac{1}{x^c}$ for all $x > N_c$. Therefore, by computing the absolute value of the difference between $Prob'(A(\vec{y}))$ and $Prob(A(\vec{y}))$, we can get the equation as follow:

$$\delta(M) = (2^{L-2})^{-M}\left(1 - \frac{1}{2^M}\right). \tag{5}$$

Therefore, we can prove that $\delta(M)$ is a negligible function based on the computational indistinguishable definition. Since M is no less than 1, $\delta(M) < (2^{L-2})^{-M}$. Then, we can let $a = 2^{L-2}$ and $\delta(M) = a^{-M}$. To find out the N_c which make $\delta(M)$ is a negligible function. we assume that $\delta(M) < M^{-c}$ where c can be any positive real integer. By taking natural logarithms, $\delta(M) < M^{-c}$ may be written as:

$$\frac{M}{\ln M} > \frac{c}{\ln a}.$$

Since the Taylor expansion of the exponential function [32] shows that $e^y > 1 + y + \frac{1}{2}y^2$ for all $y > 0$. Thus, we can get the formulation that $e^y/y > 1/y + 1 + \frac{1}{2}y$, which implies $e^y/y > \frac{1}{2}y$. Then, by replacing $\ln M$ with y, we can get the inequality as follow:

$$\frac{M}{\ln M} > \frac{\ln M}{2} \quad (M > 1).$$

Obviously, when N_c is no less than $\exp\frac{2c}{\ln a}$, $\delta(M)$ satisfies the negligible function definition. Consequently, we prove that the access patterns in Loco-ORAM is computationally indistinguishable compared with Path ORAM. Moreover, the A($\vec{y}$) in our design is still computationally indistinguishable from a random sequence of bit strings. It is because the A($\vec{y}$) in Path ORAM is computationally indistinguishable with a random sequence of bit strings. Thus, based on the transitivity/triangle inequality property in computational indistinguishability theory [33], the A($\vec{y}$) in our design is also computationally indistinguishable compared with a random sequence of bit strings. Therefore, the security of the Loco-Store can keep the same security definition of ORAM technology.

Stash Size. Although the Loco-Store holds the relevant blocks in the stash, the location-based eviction is still a greedy process for the irrelevant blocks from the stash to the server storage similar to the Path ORAM. Thus, we can calculate the upper bound of the stash size from two parts. The first part is for the usage size about relevant blocks, while the other is the usage of the irrelevant blocks. In the worst case, the three queues are filled with relevant blocks.

TABLE 2
Traces

Name	Read:Write Ratio	Read Number	Write Number	Real Block Number
INS	6.31	35642	5646	18639
WEB-Image	11.62	33664	2897	31298
FIU-Mail	0.00002	3896	41525	29716
FIU-Homes	0.27	11448	42134	23657

Then, the upper bound about this part is $\mathcal{O}((S_r + S_s) * \frac{L*Z}{4})$ where the S_r denotes the maximum length of the removed queue, the S_s denotes the maximum length of the spatial queue. since $S_r \ll N$ and $S_s \ll N$. the $\mathcal{O}((S_r + S_s) * \frac{L*Z}{4})$ can be reduced to $\mathcal{O}(logN)$.

The upper bound about the second part is similar to the Path ORAM [3]. It is because the locality-based eviction in Loco-Store divides the ORAM tree as two subtrees. Each access on the subtrees is the same as the access in a Path ORAM. Although the relevant block is held in the stash, the usage for the irrelevant part is greedily evicted from stash to server storage. Thus, we can denote the upper bound of the second part is $\mathcal{O}((logN)h(N)$ by any function of $h(N) = \omega(1)$. What's more, the replacement policy in the stash controller and the locality-based eviction scheme makes the irrelevant blocks are greedily evicted to the server storage similar to the Path ORAM. Thus, it can hardly induce stash overflow. It is noted that the superblock size does not affect the computational indistinguishability property in Loco-Store since the blocks in a superblock are randomly distributed in a path.

Correctness. To achieve the data correctness, the Loco-Store utilize the write-back policy. In other words, for each data modification operation, the updated block is written in the stash and hold by the stash controller. Only when the block is evicted, the modified data can be written to backend storage. What's more, the access algorithm can ensure that each access is to read data from the stash. The user only accesses the remote storage if the target data cannot be found in the stash. Thus, this ensures that each access data is the latest data, no dirty data. Moreover, whole blocks in the stash are stored at the buffer, and the three queues maintain the temporal and spatial locality based on the reference. Thus, the data can be consistent in the three queues simultaneously. The disadvantage of this policy is that the stash's data may lose once the system power down. However, since the proxy in our threat model, such as the organization gateway, is almost non-outage, we can practically ignore this disadvantage.

6 Experiments

6.1 Methodology

To evaluate the Loco-Store, we analyze experimental results from two main metrics, including the overall network bandwidth cost and the overall access time consumption. In Oblivious Storage, the overall access time consumption includes the time cost of the read and write operations to remote cloud storage. And each target block access causes a read and write operations for the same block number. Thus, we do not divide the overall access time into two parts: read and write. Both the servers and the trusted proxy are equipped with an Intel(R) Core(TM) i7-4790 @3.60 GHz 8 core CPU, 16 GB RAM, a 500 GB 5400 rpm hard disk and is connected to a 100 Mbps network. The installed OS is Ubuntu 16.10 LTS 64-bit System. We use another machine with the same configuration to feed the requests from clients. These high-performance servers are formed as a cluster to simulate the real cloud storage scenario.

There are four baseline designs for comparisons. The first one is the oblivious storage by directly utilizing the Path ORAM [3] technology between the trusted proxy and the cloud server called Path-Store. The second and third is the oblivious storage by integrating the Ring ORAM [11] and PrORAM [15] which is called Ring-Store and Pr-Store, respectively. Another is the state-of-the-art oblivious storage, TaoStore [9], which is also based on the trusted proxy model. We conduct the experiments on four traces. All of these traces, which are shown as Table 2, belong to real-world trace. Instructional Workload(INS) comes from a collection from a group consisting of 20 machines located in labs for undergraduate classes, which shows a robust temporal locality. Web-Image [10], [34], [35] came from a collection of the webserver which maintains a database of images and shows a strong spatial locality. This server received approximately 2,300 accesses per day. The Trace of FIU-Homes and FIU-Mail [36] are from two different applications, namely research group activities, and mail servers in FIU, respectively.

6.2 Numerical Results and Discussions

First, to explore the relationship between bandwidth cost and the maximum real block number in a bucket, Z, we conduct the experiment on Path-Store, TaoStore, Ring-Store, Pr-Store, and Loco-Store under different Z value. Moreover, previous work [13] showed that Z=3 provides the best performance in ORAM technology. Thus, Fig. 5 shows the result by sweeping the Z value from 3 to 4 while the other experimental parameters are fixed. Correctly, the stash size is fixed as 6 MB, where the maximum block number in the temporal queue is 128, and the maximum superblock number in the removed queue is 64. Moreover, the maximum superblock number of the spatial queue in Loco-ORAM is no more than the maximum block number in the temporal queue. To facilitate the description, the M_ZX in the figure means that M is the comparison method such as Path-Store, TaoStore, Ring-Store, Pr-Store, or Loco-Store, and X denotes the value of Z.

As shown in Fig. 5, the overall network bandwidth cost of all methods is swelling as the increase of the Z values under various workloads. Specifically, when Z equals 4 in FIU-Homes workload, the bandwidth cost of Loco-Store is 25.1 percent more than the bandwidth cost of Loco-Store in Z=3. Furthermore, there are similar trends in various traces. It is because both Loco-Store and the other baselines achieve

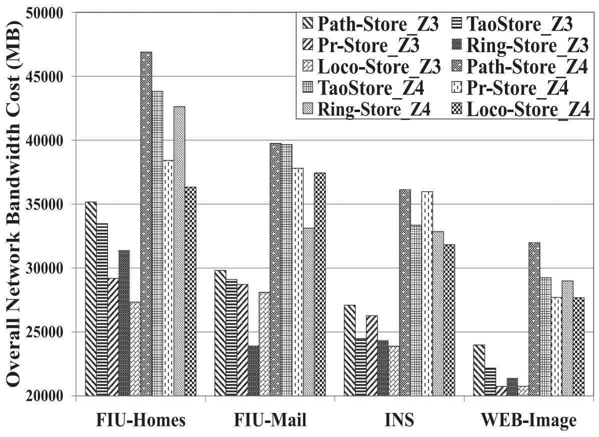


Fig. 5. The overall network bandwidth cost under various Z values.

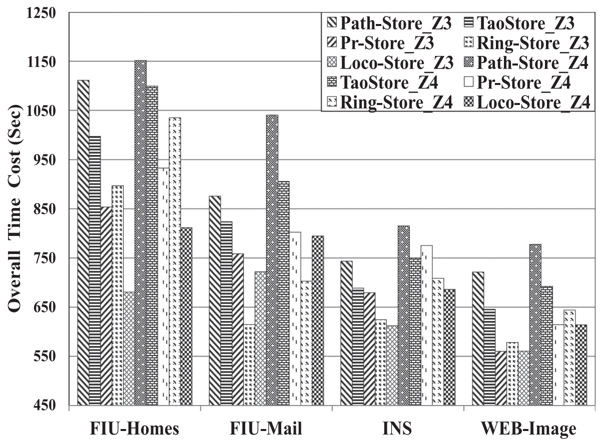


Fig. 6. The overall time cost under various Z values.

the oblivious purpose by fetching additional blocks in each access. Once stored real blocks in a bucket are less than Z, dummy blocks can pad the bucket's rest space. Thus, each access induces a higher overall network bandwidth cost for the larger Z.

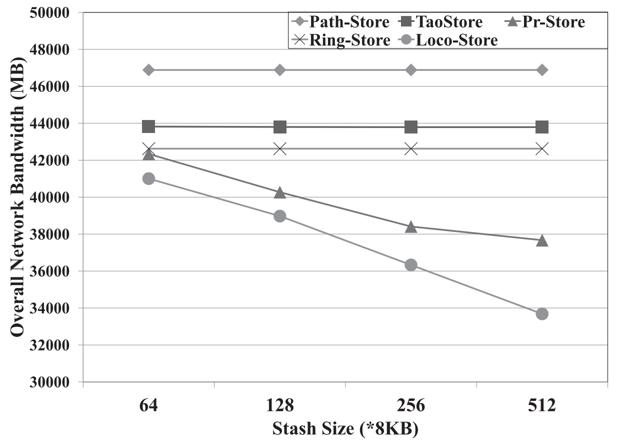
Fig. 5 shows that the overall network bandwidth cost in the Loco-Store is far less than the other baselines under the traces with a robust locality. For example, although the overall network bandwidth of TaoStore in FIU-Homes is 1.65 GB less than Path-Store when Z equals 3, Loco-Store can save 18.32 percent of the overall network bandwidth based on the TaoStore. Moreover, the Loco-Store can save 14.76 and 6.32 percent of the overall network bandwidth based on the Ring-Store and Pr-Store, respectively. Meanwhile, Loco-Store can save 22.24 percent of the overall network bandwidth based on the Path-Store. Besides, we also observer the overall time cost under different Z values. Fig. 6 list out the corresponding overall time cost to Fig. 5 and the overall time cost of Loco-Store is less than other methods except for FIU-Mail. Namely, the total time for Loco-Store in FIU-Homes is 26.17, 13, and 21.58 percent less than the TaoStore's, Pr-Store's, and Ring-Store's total time when Z equals to 4, respectively. It is because there exists a robust temporal or spatial locality in these traces, and Loco-Store can avoid unnecessary oblivious access by maintaining the temporal and spatial locality in the stash controller. Nevertheless, the merge and break process in Pr-Store is not suitable for the temporal locality. Therefore, Loco-Store can have less bandwidth cost and execution time under the trace, which has a robust temporal locality.

Moreover, only 3 percent of the overall network bandwidth is saved compared with TaoStore under the FIU-Mail trace in Fig. 5. In this trace, the Ring-Store can save more bandwidth compared with Loco-Store. It is because that too many new writing operations lead to the FIU-Mail trace has a weak locality compared with the FIU-Homes trace. Too many new writing operations, existing in FIU-Mail trace, can hardly give full play to the advantage of the locality property. Besides, since Web-Image trace shows a strong spatial locality and INS trace shows a robust temporal locality, the Loco-Store can avoid more oblivious access by the block hits. It is noted that the overall network bandwidth cost between Loco-Store and Pr-Store is similar in Web-Image trace. Since the Web-Image trace only shows a strong spatial locality. Thus, the benefits of Loco-Store from the temporal locality in Web-Image trace becomes little.

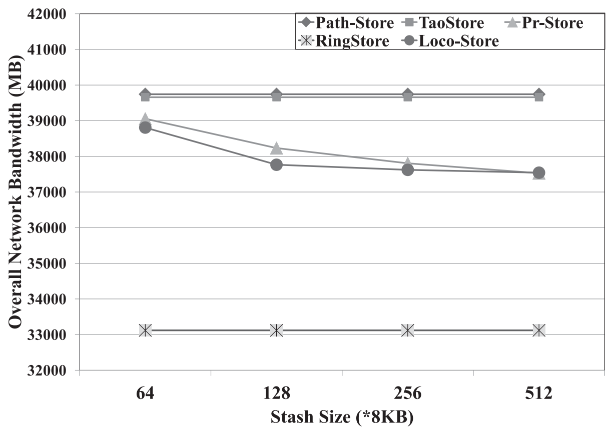
To further explore influence by different parameters setting, we also conduct the experiments under four real traces by changing the maximum length of the temporal queue. As we discussed in Section 5, the stash size in Loco-Store is related to the maximum length of the temporal queue and removed queue. However, there is no queue setting in TaoStore, Path-Store, Ring-Store, and Pr-Store. Nevertheless, when we fixed other parameter settings in Loco-Store, each maximum length setting of the temporal queue corresponds to a stash size. For example, when Z equals to 4, and each block is 8 KB, the stash size of Loco-Store is from 2 MB to 16 MB according to the maximum length of the temporal queue from 32 to 128. Thus, as shown in Fig. 7, we compare the experimental results under various stash size. It shows that the overall network bandwidth cost in Path-Store, TaoStore, Ring-Store is almost unchanged as the increase of Stash Size. Pr-Store only indicates a similar trend with Loco-Store under the trace with a strong spatial locality such as WEB-Image. Moreover, Loco-Store shows a better performance in the traces which has robust temporal or spatial locality.

Specifically, the overall network bandwidth cost of Loco-Store is less than other baselines, especially in the FIU-Homes, INS, and WEB-Image traces. For example, as shown in Fig. 7a, the overall network bandwidth cost of TaoStore is 39.19 percent more than the overall network bandwidth cost of Loco-Store when the stash size is 512*8KB. Moreover, the network bandwidth decreased by Loco-Store is more than that by TaoStore and Path-Store as the increasing of stash size. The main reason is that the Path-Store and the TaoStore induce much more remote accesses than the Loco-Store. As we known, each oblivious accesses from server storage causes lots of useless block transmissions in Path-Store and TaoStore. The Loco-Store can decrease the oblivious accesses number by embedding the relevant blocks into each oblivious access.

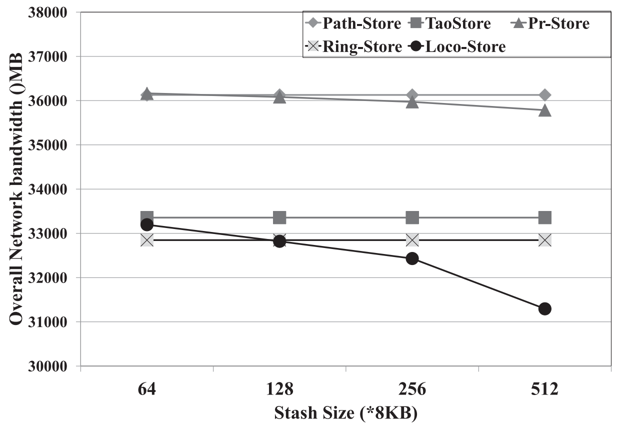
As shown in Fig. 7, the network bandwidth of the Path-Store, TaoStore, and Ring-Store are constant with the increasing of the stash size. It is because the purpose of these methods is to keep the stash occupancy low. The change of stash size has little effect on network bandwidth cost. On the contrary, both Loco-Store and Pr-Store attempt



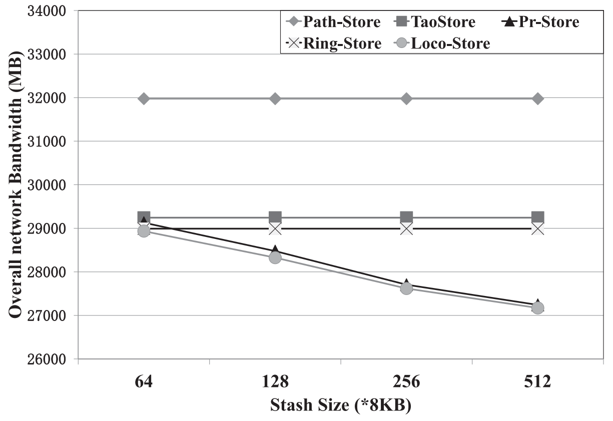
(a) FIU-Homes



(b) FIU-Mail



(c) INS



(d) WEB-Image

Fig. 7. The overall network bandwidth under different stash size.

to hold some related blocks to avoid unnecessary oblivious access. Thus, the overall network bandwidth cost of our method and Pr-Store changed under various traces with a robust locality. However, although the performance of the Loco-Store still outperforms other baselines, the saving bandwidth cost of the Loco-Store becomes less as the stash size increases in the FIU-Mail trace. It is because the FIU-Mail trace shows weak locality because of too many write new block operations. Thus, the Ring-Store can save more bandwidth than Loco-Store under the trace with a weak locality such as FIU-Mail. And the Loco-Store can obtain higher benefits under trace with a robust locality.

Besides the maximum length of the temporal queue, we also explore the effect of the parameters in other queues. Since the remove queue and spatial queue in the stash controller of Loco-Store are used to hold relevant blocks with the temporal queue, the experimental effect of changing the maximum length of remove queue or spatial queue is similar to the results under various temporal queue. Thus, we do not repeatedly list the results by changing the maximum length of the spatial queue or removed queue. However, the parameter $remove\_size$ in the locality-based eviction process of the Loco-Store can also impact the performance. Therefore, we calculate the saving network bandwidth by changing the $remove\_size$ under various workloads. First, we initial overall network bandwidth cost when the $remove\_size$ is set as zero. Then, we also count the saving bandwidth cost by compared with the initial overall network bandwidth cost by changing the $remove\_size$ from 64 to 256. Fig. 8 shows the experimental results under various traces. As the increase of the $remove\_size$, the saving network bandwidth cost in Loco-Store is also augmented. It is because the remove queue not only works as a buffer to support the writing the superblock based on spatial locality but also reduce the remote oblivious access number by increasing the block hit in the stash. Specifically, the locality-based eviction scheme can only evict the

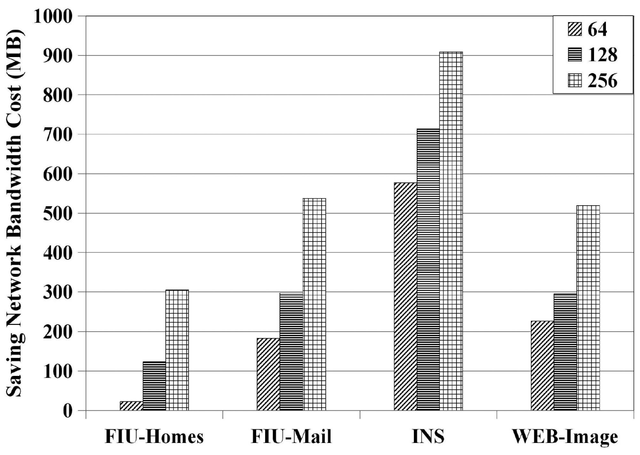


Fig. 8. The saving network bandwidth of the loco-store under various $remove\_size$ (*8KB).

real blocks into the cloud once the superblock number of the remove queue exceeds the $remove\_size$ setting.

7 Conclusion

In this paper, we analyze ORAM technology and present its limitations in Oblivious Data Storage. Traditional ORAM technology can hardly align successive blocks together and retain hot blocks for future accesses. Then, inspired by the successful adoption of utilizing locality in storage domains, we propose a novel locality-based Oblivious Data Storage, Loco-Store, to integrate the features of locality and obliviousness in the cloud scenario. In Loco-Store, a novel stash controller taking advantage of both temporal and spatial locality is designed, and a locality sensitive eviction scheme is also proposed to keep the provable security compared with ORAM. Finally, Based on theoretical proof and the conducted experimental results, Loco-Store outperforms the state-of-the-art Oblivious Data Storage solution, TaoStore. It can save the network bandwidth up to 39.19 percent, and reduce the overall access time by 26.17 percent.

In the future, we try to further evaluate the Loco-Store by deploying onto Google Drive or Amazon EC2 and exploring the theoretical setting method of the superblock size in Loco-Store. Furthermore, the situation becomes more complex and practical when considering access control and concurrency problems in cloud storage. Thus, we attempt to further add these two features based on Loco-Store and defense the information leakage through timing channels such as when or how frequently the client makes data requests. Although many researchers are dedicated to improving ORAM technology's performance, it can hardly have the same performance as the non-secure system, not even Loco-Store. Hence, we will rethink the oblivious technology.

Acknowledgments

This work was supported by the National Key Research and Development Program of China under Grants 2016YFB0800402 and 2016QY01W0202, National Natural Science Foundation of China under Grants U1836204, U1936108, 61572221, 61433006, and U1401258, Teaching Reform Foundation under Grant 192JXJ129, South of China University research foundation under Grant 190XQD121, Hunan Province research foundation under Grant 19C1624, and the Ministry of Education Humanities and Foundation on Humanities and Social Sciences under Grant 20YJC880027. This work is also supported in part by the National Science Foundation under Grant CNS 1526190.

References

[1] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *Proc. 19th Annu. Netw. Distrib. Syst. Secur. Symp.*, 2012, pp. 237–252.
[2] O. Goldreich, "Towards a theory of software protection and simulation by oblivious rams," in *Proc. 19th Annu. ACM Symp. Theory Comput.*, 1987, pp. 182–194.
[3] E. Stefanov *et al.*, "Path ORAM: An extremely simple oblivious RAM protocol," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 299–310.
[4] R. Wang, Y. Zhang, and J. Yang, "D-ORAM: Path-oram delegation for low execution interference on cloud servers with untrusted memory," in *Proc. IEEE Int. Symp. High Perform. Comput. Architect.*, 2018, pp. 416–427.
[5] A. Shafiee, R. Balasubramonian, M. Tiwari, and F. Li, "Secure DIMM: Moving ORAM primitives closer to memory," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2018, pp. 428–440.
[6] X. Zhang *et al.*, "Fork path: Improving efficiency of ORAM by removing redundant memory accesses," in *Proc. 48th Int. Symp. Microarchit.*, 2015, pp. 102–114.
[7] N. Fujieda, R. Yamauchi, H. Fujita, and S. Ichikawa, "A virtual cache for overlapped memory accesses of path ORAM," *Int. J. Netw. Comput.*, vol. 7, no. 2, pp. 106–123, 2017.
[8] E. Stefanov and E. Shi, "Oblivistore: High performance oblivious cloud storage," in *Proc. IEEE Symp. Security Privacy*, 2013, pp. 253–267.
[9] C. Sahin, V. Zakhary, A. El Abbadi, H. Lin, and S. Tessaro, "Taostore: Overcoming asynchronicity in oblivious data storage," in *Proc. IEEE Symp. Security Privacy*, 2016, pp. 198–217.
[10] J. Wang and Y. Hu, "WOLF - A novel reordering write buffer to boost the performance of log-structured file systems," in *Proc. Conf. File Storage Technol.*, 2002, pp. 47–60.
[11] L. Ren *et al.*, "Ring ORAM: Closing the gap between small and large client storage oblivious RAM," *IACR Cryptol. ePrint Archive*, vol. 2014, 2014, Art. no. 997.
[12] C. W. Fletcher, M. Naveed, L. Ren, E. Shi, and E. Stefanov, "Bucket ORAM: Single online roundtrip, constant bandwidth oblivious RAM," *IACR Cryptol. ePrint Archive*, vol. 2015, 2015, Art. no. 1065.
[13] L. Ren, X. Yu, C. W. Fletcher, M. van Dijk, and S. Devadas, "Design space exploration and optimization of path oblivious RAM in secure processors," in *Proc. 40th Annu. Int. Symp. Comput. Archit.*, 2013, pp. 571–582.
[14] R. Wang, Y. Zhang, and J. Yang, "Cooperative path-oram for effective memory bandwidth sharing in server settings," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2017, pp. 325–336.
[15] X. Yu *et al.*, "Proram: Dynamic prefetcher for oblivious RAM," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, 2015, pp. 616–628.
[16] S. Devadas, M. van Dijk, C. W. Fletcher, L. Ren, E. Shi, and D. Wichs, "Onion ORAM: A constant bandwidth blowup oblivious RAM," in *Proc. 13th Int. Conf. Theory Cryptogr.*, 2016, pp. 145–174.
[17] T. Mayberry, E. Blass, and A. H. Chan, "Efficient private file retrieval by combining ORAM and PIR," in *Proc. 21st Annu. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 778–789.
[18] D. Apon, J. Katz, E. Shi, and A. Thiruvengadam, "Verifiable oblivious storage," in *Proc. 17th Int. Conf. Practice Theory Public-Key Cryptogr.*, 2014, pp. 131–148.
[19] T. Moataz, T. Mayberry, and E. O. Blass, "Constant communication oram with small blocksize," in *Proc. ACM Sigsac Conf. Comput. Commun. Secur.*, 2015, pp. 862–873.
[20] J. Trostle and A. Parrish, "Efficient computationally private information retrieval from anonymity or trapdoor groups," in *Proc. Int. Conf. Inf. Secur.*, 2010, pp. 114–128.
[21] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Int. Conf. Theory Appl. Cryptogr. Techn.*, 1999, pp. 223–238.
[22] E. Shi, T. H. H. Chan, E. Stefanov, and M. Li, "Oblivious ram with O((log N) 3) worst-case cost," in *Proc. Int. Conf. Theory Appl. Cryptology Inf. Secur.*, 2011, pp. 197–214.
[23] J. Dautrich and C. V. Ravishankar, "Combining ORAM with PIR to minimize bandwidth costs," in *Proc. 5th ACM Conf. Data Appl. Secur. Privacy*, 2015, pp. 289–296.
[24] E. Stefanov, E. Shi, and D. X. Song, "Towards practical oblivious RAM," in *Proc. 19th Annu. Netw. Distrib. Syst. Secur, Symp.*, 2012, pp. 210–229.
[25] T. Hoang, C. D. Ozkaptan, A. A. Yavuz, J. Guajardo, and T. Nguyen, "S^3ORAM: A computation-efficient and constant client bandwidth blowup ORAM with shamir secret sharing," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 491–505.
[26] C. Nagarajan, A. Shafiee, R. Balasubramonian, and M. Tiwari, "ρ: Relaxed hierarchical ORAM," in *Proc. 24th Int. Conf. Architect. Support Program. Languages Operating Syst.*, 2019, pp. 659–671.
[27] H. Cao, R. Li, W. Tian, Z. Xu, and W. Xiao, "Blockchain-based accountability for multi-party oblivious RAM," *J. Parallel Distrib. Comput.*, vol. 137, pp. 224–237, 2020.
[28] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems," in *Proc. 16th Annu. Int. Cryptology Conf. Advances Cryptol.*, 1996, pp. 104–113.

[29] H. Jingmin and L. Kaicheng, “A new probabilistic encryption scheme,” in *Proc. Advances Workshop Theory Appl. Cryptogr. Techn.*, 1988, pp. 415–418.

[30] E. J. O’Neil, P. E. O’Neil, and G. Weikum, “The LRU-K page replacement algorithm for database disk buffering,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1993, pp. 297–306.

[31] D. Lee *et al.*, “LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies,” *IEEE Trans. Comput.*, vol. 50, no. 12, pp. 1352–1361, Dec. 2001.

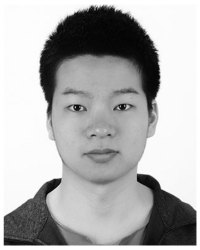
[32] Wikipedia contributors, “Taylor series — Wikipedia, the free encyclopedia,” 2020. Accessed: Jul. 2, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Taylor_series&oldid=964429633

[33] O. Goldreich, *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[34] E. Lee, S. Yoo, and H. Bahn, “Design and implementation of a journaling file system for phase-change memory,” *IEEE Trans. Comput.*, vol. 64, no. 5, pp. 1349–1360, May 2015.

[35] D. Roselli, J. R. Lorch, and T. E. Anderson, “A comparison of file system workloads,” in *Proc. Conf. Usenix Tech. Conf.*, 2000, pp. 4–4.

[36] R. Koller and R. Rangaswami, “I/O deduplication: Utilizing content similarity to improve I/O performance,” *ACM Trans. Storage*, vol. 6, no. 3, pp. 1–26, 2010.



Wenlong Tian (Student Member, IEEE) received the MS and PhD degrees from the School of Computer Science and Technology, Huazhong University of Science and Technology, in 2015 and 2019, respectively. Currently he is currently an assistant professor with the School of Computer Science and Technology, the University of South China. His research interests include cloud computing, system security, and big data management.

Ruixuan Li (Member, IEEE) received the BS, MS, and PhD degrees from the School of Computer Science and Technology, Huazhong University of Science and Technology, in 1997, 2000, and 2004 respectively. He is currently a professor with the School of Computer Science and Technology, the Huazhong University of Science and Technology, and is the director of the Intelligent and Distributed Computing Laboratory. His research interests include cloud computing, big data, and system security.

Zhiyong Xu (Senior Member, IEEE) received the BS and MS degrees in computer science and engineering from the Huazhong University of Science and Technology, China, in 1994 and 1997, respectively, and the PhD degree in computer engineering from the University of Cincinnati, in 2003. He is currently an associate professor with the Department of Mathematics and Computer Science, Suffolk University. His research interests include Cloud Computing, Cloud Security, High performance I/O and File systems, and Parallel and Distributed Computing.

Weijun Xiao (Senior Member, IEEE) received the BS and MS degrees in computer science from the Huazhong University of Science and Technology, China, in 1995 and 1998, respectively, and the PhD degree in electrical engineering from the University of Rhode Island, Kingston, RI. He is currently an associate professor with the Department of Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, VA. His research interests include computer architecture, networked storage system, embedded system, and performance evaluation.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.