IARJSET

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

NCAIT 2017

JSS Academy of Technical Education
Vol. 4, Special Issue 8, May 2017

Privacy-Preserving Access to Big Data in Cloud and Load balancing using ORAM Algorithm

Swathi B R

Student, RNSIT, Bangalore

Abstract: In the era of big data, many users and companies start to move their data to cloud storage to simplify data man-agement and reduce data maintenance cost. HoIver, security and privacy issues become major concerns because third-partycloud service providers are not always trusty. Although data contents can be protected by encryption, the access patterns that contain important information are still exposed to clouds or malicious attackers. In this paper, I apply the ORAM algorithm to enable privacy-preserving access to big data that are deployed in distributed file systems built upon hundreds or thousands of servers in a single or multiple geo-distributed cloud sites. Since the ORAM algorithm would lead to serious access load unbalance among storage servers, I study a data placement problem to achieve a load balanced storage system with improved availability and responsiveness. Due to the NP-hardness of this problem, I propose a low-complexity algorithm that can deal with large-scale problem size with respect to big data. Extensive simulations are conducted to show that my proposed algorithm finds results close to the optimal solution, and significantly outperforms a random data placement algorithm.

I. INTRODUCT ION

Big data has emerged in various domains including science, engineering and commerce. For example, the amount of photos currently stored by Facebook is over 20 petabytes, and it continues to grow with 60 terabytes each Iek [1]. In the era of big data, cloud becomes a perfect candidate for data storage by providing virtually unlimited storage that can be accessed over network. By outsmycing large volumes of data to cloud storage, such as Google Drive, Dropbox and Amazon S3, users can simplify their data management and reduce data maintenance cost due to the pay-as-you-use model. HoIver, some users and companies still hesitate to move their data to cloud because of security and privacy concerns. Although encryption can protect the data confidentiality, it is insufficient because access patterns can also leak important information. For instance, over 80% of encrypted email queries can be identified according to access pattern in [2].
The access privacy problem is first addressed by private information retrieval (PIR) technique [3] that allows a user to retrieve a block from a database of N items held by a server that learns nothing about this block. Unfortunately, Sion et al. [4] have shown that existing PIR schemes will never be more efficient than a trivial PIR scheme of downloading the entire database. The extremely poor performance of PIR makes it inapplicable in cloud storage with big data.

Oblivious RAM (ORAM) is later proposed to hide data access privacy with improved performance. Its basic idea isto periodically reshuffle data blocks stored in an untrusted server such that user access cannot be tracked. Goodrich et al.
[5] have proposed an ORAM algorithm with $O(\sqrt{N})$ clientstorage to achieve $O(\log N)$ amortized cost, i.e., each oblivious read or write leads to $O(\log N)$ data access operations on average. Shi et al. [6] further reduce the client storage to $O(1)$.

In this paper, I apply the ORAM algorithm to enable privacy-preserving access to big data in clouds. To deal with the challenge of accommodating huge volume of data that continuously grows in high velocity, big data are stored in distributed file systems built upon hundreds or thousands of servers in a single or multiple geo-distributed cloud sites. When ORAM is directly applied on such distributed storage systems, I observe that even if all data blocks are evenly accessed by users, access load on servers would be seriously unbalanced, i.e., lots of data access requests are sent to several servers, but only a few to others. The servers with high load are apt to be system bottleneck or failure points in the system. Motivated by this observation, I exploit the data access characteristics of ORAM, and propose a data placement algorithm to achieve load balance, thus improving overall system availability and responsiveness.

The main contributions of this paper are summarised as fol-lows. First, I study the privacy-preserving data access to big data in an untrusted cloud by applying the ORAM algorithm. In conjunction with encryption, ORAM-based solutions can hide not only data contents but also access patterns from third-party cloud service provider and malicious attackers. Second, I investigate a load balance problem in applying ORAM on distributed file systems. This problem is proved to be NP-hard, and formulated as a mixed integer linear programming (MILP) problem. I propose a low-complexity algorithm that can deal with large-scale problem instances with respect to big data. Third, extensive simulations are conducted to show that the performance of my proposed algorithm is close to the optimal solution, and significantly outperforms a random data placement algorithm.

The rest of this paper are organized as follows. I review important related work in Section II. Section III presents some necessary preliminaries about ORAM algorithm and my motivation. The system model and problem formulation are given in Section IV, folloId by an efficient algorithm proposed in Section V. I show extensive simulation results in Section VI. Section VII finally concludes this paper.

II. RELATED WORK

A. Cloud storage

Cloud storage has attracted a lot of attentions from both industry and academic. Many Ill-known cloud service providers have started their cloud storage services during past few years, such as Microsoft SkyDrive, Amazon S3, and Google Drive. RAID (Redundant Array of Inexpensive Disks) technique is integrated in HAIL [7] that manages remote file integrity and availability across a collection of servers. Similarly, Dabek et al. [8] use RAID-like techniques to ensure the availability and durability of data in distributed systems. To improve the reliability and security of cloud storage, Bessani et al. [9] have proposed a distributed storage system called DEPSKY that integrates encryption, encoding and replication. IRIS [10] is proposed as an authenticated file system that lets enterprises store data in the cloud with resilience against potentially untrusted cloud providers. There are several pro-posals dealing with data availability by constructing distributed storage systems across several cloud sites. Wu et al. [11] have proposed SPANStore, a key-value storage system that exports a unified view of storage services in geo-graphically distributed data centers. It minimizes an application provider's cost with three key techniques, i.e., exploiting pricing discrep-ancies across providers, estimating application workload at the right granularity, and minimizing the usage of computational resmyces.

B. Oblivious RAM

As originally proposed by Goldreich and Ostrovsky [12], ORAM allows a trusted processor to use an untrusted RAM. Most existing ORAM solutions use the basic memory struc-ture suggested by Ostrovsky's Hierarchical Scheme [13]. The ORAM is arranged in a series of progressively larger caches. Each cache consists of a hash table of buckets. When a block is requested, the algorithm checks a bucket at each level of the hierarchy. If the block is found, the search continues for a dummy block such that the location of his desired block is hidden. Finally, the block is reinserted into the top-level cache. When a cache is close to overflowing, it is obliviously shuffled into the cache below it.

Recent ORAM work has explored optimisations of the classic Hierarchical Scheme [13], [14], including the use of cuckoo hashing [15] and Bloom filters [16]. Williams et al. [17] have presented SR-ORAM as the first single-round-trip polylogarithmic time ORAM that requires only logarithmic client storage. Taking only a single round trip to perform a query, SR-ORAM has an online communication/computation cost of O(log n log log n). Lorch et al. [18] have presented Shroud, a general storage system that hides data access pat-terns from the servers. Shroud uses many secure coprocessors acting in parallel as client proxies in the data center.

III. PRELIMINARIES AND MOTIVATION

In this section, I first present some necessary background about ORAM, and then show the load unbalance phenomenon that motivates my proposal.

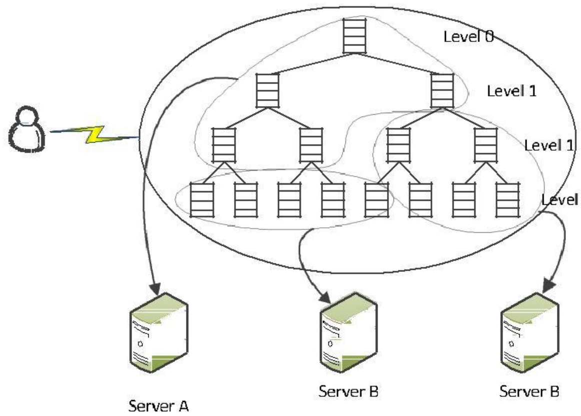

Fig. 1. ORAM-based cloud storage.

A. The ORAM algorithm

I consider a client that would like to store and retrieve its big data in cloud that is honest but curious. In other words, the cloud cannot tamper with or modify the data, but could learn information about the data. The data are divided into blocks, each of which is identified by a unique address. For example, a typical value of block size is 64KB or 256KB. Data stored on cloud are organized as a tree, where each node is referred to as a bucket that stores several data blocks. An example of a binary tree structure is shown in Fig. 1. Note that any arbitrary tree structure is applicable in ORAM. Following the work in [6], I translate each read or write operation into two primitives ReadAndRemoveand Add that are definedas follows.

ReadAndRemove(u): given an address u specifiedby theclient, the cloud returns the corresponding data block, and removes it from storage.
Add(u, d): the client writes block d to address u at the clientstorage.

With above two primitives, each read(u) operation can be replaced by a ReadAndRemove(u)folloId by an Add(u, d) that writes the same data block back to address u. Similarly, to implement a write(u, d) operation, I conduct a redundant ReadAndRemove(u) before Add(u, d).

IARJSET

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

NCAIT 2017

JSS Academy of Technical Education

Vol. 4, Special Issue 8, May 2017

Although the numberof access operations are doubled in ORAM, it prevents the untrusted cloud from distinguishing read and write operations. The implementation of ReadAndRemove(u) and Add(u, d) is critical for hiding access patterns in ORAM. When a datablock is written into the cloud storage, it is always inserted into the root bucket in the level 0 as shown in Fig. 1. As more data blocks are being added in the root bucket, it will eventually be full without residual capacity to accommodate new blocks. To avoid overflowing, data blocks in each non-leaf bucket are periodically evicted to its children buckets. I assign a random number called designator to each newly added data block to indicate which leaf bucket it is evicted to along the tree. Note that only the client knows the mapping betIen block address and its associated designator. At each level of tree, the client randomly chooses several buckets to evict. In order to prevent the cloud from tracking the eviction process, dummy blocks are inserted into other children buckets that do not receive the real data block.

To read a data block, the client first looks up its correspond-ing designator in local storage, and then reads all buckets along the path betIen the root and the leaf bucket indicated by this designator in the tree. When the data block is found, I remove it from its current bucket, and write it back to the root bucket with a new designator. In such a way, the cloud cannot infer which block is read because repeated reads for the same block will produce different lookup paths through the tree.

B. Motivation

To deploy an ORAM-based storage in a distributed system, I need to partition the corresponding tree structure into multiple parts, each of which is stored in a server. For example, I consider to store the ORAM tree shown in Fig. 1 into three servers, each of which can accommodate at most 5 buckets. A partition scheme is shown in Fig. 1. Since the root bucket is accessed in each read and write operation, the server A holding the root bucket has the highest access load. On the other hand, each read operation only involves one leaf node, leading to the loIst load on server B that stores five leaf nodes in level 3. From this example, I observe that ORAM-based storage would lead to serious unbalanced data access load among servers without a delicate bucket placement, which motivates us to develop an algorithm to optimize the data placement in next section.

IV. SYSTEM MODEL AND PROBLEM FORMULATION

I consider to deploy an ORAM-based storage with n buckets of size B to m servers residing in multiple clouds. Some authorized users generate a set of access requests that have been translated into a serious of ReadAndRemove(u) and Add(u, d) operations. Each bucket is the minimum storageunit, and is associated with an access rate a_i due to the read, write, and eviction operations in ORAM. Note that the access rate of each bucket can be estimated according to the characteristics of the ORAM algorithm, such as tree structure, and eviction probability. The i-th server can accommodate at most C_i buckets. Based on the system model, my load balance problem can be described as a max-min problem as follows.

Definition 1: The problem of load balance for deployingORAM-based storage in clouds (LBOC): given a tree-based ORAM structure, and a set of storage servers, the LBOC problem seeks a data placement such that the maximum access load among all servers is minimized.

Since a bucket is the minimum access unit in ORAM, I define binary variables x_{ij} to describe bucket placement as follows.

$$x_{ij} = \begin{cases} 1, & \text{if the i-th bucket is placed on the j-th server,} \\ 0, & \text{otherwise.} \end{cases}$$

I also define a variable y_j to represent the total access rate in the j-th server, and the LBOC problem can be formulated as a mixed integer linear programming (MILP) as follows.

$$\min Y$$

$$y_j \le Y, \forall 1 \le j \le m; \quad (1)$$

$$y_j = \sum_{i=1}^{N} a_i x_{ij}, \forall 1 \le j \le m; \quad (2)$$

$$\sum_{j=1}^{M} x_{ij} = 1, \forall 1 \le i \le n; \quad (3)$$

$$\sum_{i=1}^{N} x_{ij} \le C_j, \forall 1 \le j \le m; \quad (4)$$

$$x_{ij} \in \{0, 1\}, \forall 1 \le i \le n, 1 \le j \le m.$$

In above formulation, variable Y denotes the maximum access rate of all servers, which is guaranteed by constraint

(1). The calculation of total access rate y_j of each server is represented by constraint (2). I impose constraint (3) because each bucket has to be placed at only one server. Finally, the capacity constraint of each server is represented by (4). In the following, I analyze the hardness of the LBOC problem by proving its NP-hardness in a formal way.

Theorem 1: The LBOC problem is NP-hard.

Proof: In order to prove an optimization problem tobe NP-hard, I need to show the NP-completeness of its decision form, i.e., I attempt to find a bucket placement such that the maximum access load is no greater than Y . It is easy to see that such a problem is in NP class as the objective

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
NCAIT 2017
JSS Academy of Technical Education
Vol. 4, Special Issue 8, May 2017

associated with a given solution can be evaluated in a polynomial time. The remaining proof is done by reducing the Ill-known 2-partition problem, i.e., given a set of numbers $S = \{s_1, s_2, ..., s_n\}$, I attempt to divide them into two sets S_1 and S2 such that $_{si}\in_{S_1} si = {}_{sj}\in_{S_2} sj$.

I now describe the reduction from the 2-partition problem to an instance of the LBOC problem. I create an ORAM storage with n buckets, each of which has an access rate $a_i = s_i$. There are two severs, each of which can store at most n

buckets. Finally, I let $Y = \frac{1}{2}$ s . It is easy to verify $s_i \in Si$

that the 2-partition problem has a solution if and only if the constructed LBO problem has a solution that satisfies the load requirements. ■

V. ALGORITHM DESIGN

Due to the NP-hardness, I design an efficient heuristic algorithm to solve the LBOC problem in this section. My basic idea is to first solve the MILP problem formulated in last section by relaxing all integer variables, and then find a feasible integer solution by rounding the results. Although the time complexity of this algorithm is polynomial, additional challenges arise in dealing with big data storage. The ORAM tree would contain a large number of buckets to accommodate big data, resulting in too many variables and constraints in the formulation. Solving such a large-scale linear programming, even with all real variables, would be time-consuming, or evenimpossible because of physical memory constraints on some computers.

To overcome this difficulty, I propose a low-complexity algorithm called ILB (Iterative Load Balancer) to iteratively place buckets on servers, such that I only need to deal with a small-scale linear programming in each iteration. The pseudo code of my algorithm is shown in the following Algorithm 1.

Algorithm 1 The ILB algorithm

$C_j^{res} = C_j, \forall 1 \le j \le m;$
$y_j^{curr} = 0, \forall 1 \le j \le m;$
while there are buckets that haven't been placed **do**
put a set of unplaced buckets in set N ;
solve the following linear programming;

min Y

$$y_j + y_j^{curr} \le Y, \forall 1 \le j \le m; \quad (5)$$

$$y_j = a_i x_{ij}, \forall 1 \le j \le m; \quad (6)$$
i∈N
m

$$x_{ij} = 1, \forall i \in N; \quad (7)$$
j=1

$$x_{ij} \le C_j^{res}, \forall 1 \le j \le m; \quad (8)$$
i∈N

$$0 \le x_{ij} \le 1, \forall i \in N, 1 \le j \le m. \quad (9)$$

sort variables x_{ij} in a descending order according to their results;
for each xijin the sorted order **do**
if the i-th bucket in N hasn't been placed andC $_j^{res}$> 0 **then**
place this bucket on the j-th server;
$C_j^{res} = C_j^{res} - 1;$
ycurrcurr

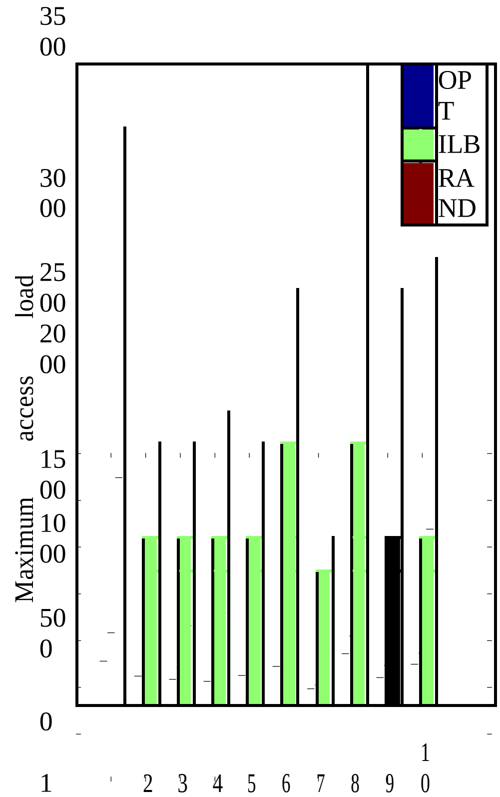

Fig. 2. Comparison with optimal solution in 10 random instances.

VI. PERFORMANCE EVALUAT ION

In this section, I study the performance of my proposed algorithm under various network settings. For comparison, I also show the performance of a random placement algorithm denoted by RAND. All simulations are conducted using Mat- lab in a computer equipped with 3.4 GHz Intel Core i7 CPU and 8G memory.

I first evaluate the performance of ILB by comparing its results with the optimal solution. I consider to deploy 200 data buckets to 10 servers, and show the results of 10 random instances in Fig. 2. In average, the performance of ILB is 1.15 times of optimal solution, while the corresponding ratio of RAND is 1.91.

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

NCAIT 2017

JSS Academy of Technical Education

Vol. 4, Special Issue 8, May 2017

j and yj residual capacity and current access load on the j-th server, respectively. After their initialization in lines 1 and 2, I conduct bucket placement in iterations in the following **while** loop. In each iteration, I consider a set of buckets N , and solve a linear programming with respect to N , current access load and C_j^{res} on each server. Different from the MILP in last section, I relax x_{ij} by letting it be a real variable betIen 0 and 1 as shown in (9). In addition, I take current access load ycurr into consideration in constraint (5), and constrain j the capacity of each server with C res in (8).After solving this linear programming, I sort variable xij in a descending order according to their results. I place the i-th bucket in set N to the server with maximum value of xij among all servers. Such placement is expected toachieve comparable performance to the optimal solution Because the real value xij would represent the probability of the corresponding optimal data placement. Finally I finish current iteration by updating the values of C resj and by averaging results over 50 random instances. The influence of number of buckets is first investigated by changing its value from 600 to 1000, and the number of servers is fixed to 10. The server capacity is randomly specified as a Gaussian distribution with mean of 100 and variance of 20. In each iteration of ILB, I consider data placement for 100 buckets.

As shown in Fig. 3, the performance of both algorithms shows as an increasing function of bucket number. Moreover, the performance gap betIen two algorithms increases as the bucket number grows. For example, when the number of buckets is 600, the maximum access rate of RAND is 17% higher. The performance gap increases to 33% as bucket number grows to 1000. The results indicate that ILB can effectively reduce the maximum access rate because of my delicate design.

I then study the effect of different variance of server capacity in instances with 1000 buckets and 10 servers. The mean value of server capacity is fixed to 100. As shown in Fig. 4, although the performance of both algorithms increases as the variance grows, their performance gap becomes small.

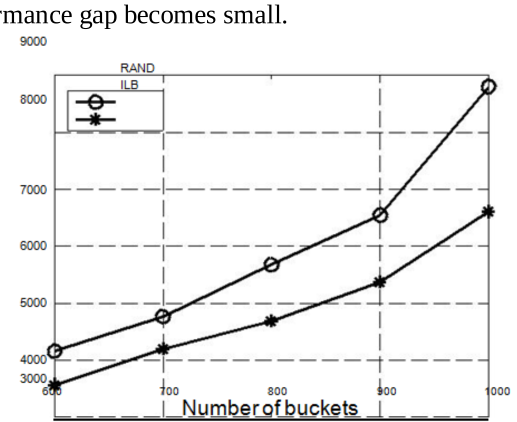

Fig. 3. Maximum access rate versus different number of buckets.

That is because the servers with small capacity quickly becomes full during data placement, while lots of buckets have to be accommodated in the servers with large capacity, which leading to high

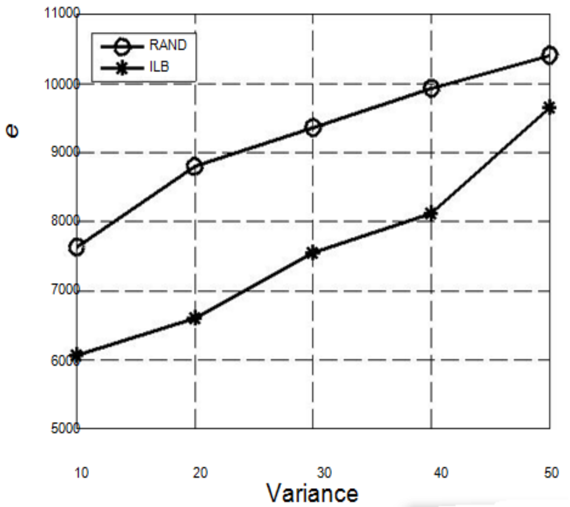

Fig. 4. Maximum access rate versus different variance of server capacity.

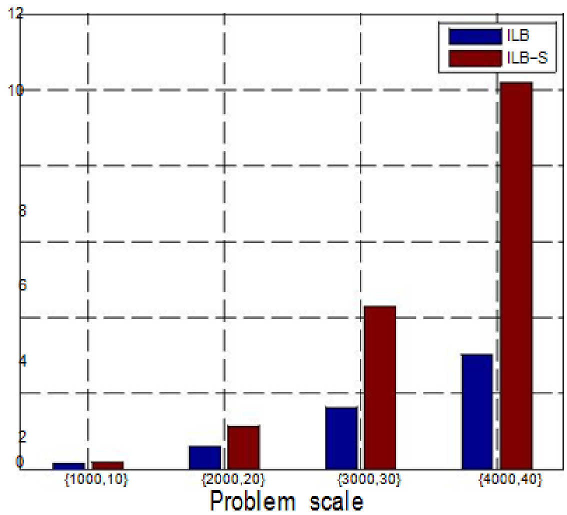

Fig. 5. Execution time under different instance scales.

VII. CONCLUSION

In this paper, I apply the ORAM algorithm to achieve privacy-preserving access to big data in clouds. I observe a load unbalance phenomenon after deploying ORAM-based storage to multiple servers, which motivates us to investigate a data placement problem to achieve load balance.

This problem is proved to be NP-hard. I propose a low-complexity algo-rithm to solve this problem with respect to large data volumes. Simulation results show that the performance of my proposed algorithm is close to optimal solution, and it outperforms a random data placement algorithm.

REFERENCES

[1] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel, "Finding a needle in haystack: Facebook's photo storage," in USENIX OSDI, 2010, pp. 1–8.
[2] M. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in Networkand Distributed System Security Symposium, 2012.
[3] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private informa-tion retrieval," Jmynal of the ACM (JACM), vol. 45, no. 6, pp. 965–981, 1998.
[4] R. Sion and B. Carbunar, "On the computational practicality of private information retrieval," in Proceedings of the Network and DistributedSystems Security Symposium, 2007, pp. 2006–06.
[5] M. T. Goodrich and M. Mitzenmacher, "Mapreduce parallel cuckoo hashing and oblivious ram simulations," CoRR, vol. abs/1007.1259, 2010.
[6] E. Shi, T.-H. H. Chan, E. Stefanov, and M. Li, "Oblivious ram with o ((logn) 3) worst-case cost," in Advances in Cryptology–ASIACRYPT 2011. Springer, 2011, pp. 197–214.
[7] K. D. BoIrs, A. Juels, and A. Oprea, "Hail: A high-availability and integrity layer for cloud storage," in Proceedings of the 16th ACMConference on Computer and Communications Security, 2009, pp. 187–198.
[8] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with cfs," in ACM Symposium on OperatingSystems Principles, 2001, pp. 202–215.
[9] A. Bessani, M. Correia, B. Quaresma, F. Andre´, and P. Sousa, "Depsky: Dependable and secure storage in a cloud-of-clouds," in Proceedings ofthe Sixth Conference on Computer Systems, 2011, pp. 31–46.
[10] E. Stefanov, M. van Dijk, A. Juels, and A. Oprea, "Iris: A scalable cloud file system with efficient integrity checks," in Proceedings of the 28thAnnual Computer Security Applications Conference, 2012, pp. 229–238.
[11] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Mad-hyastha, "Spanstore: Cost-effective geo-replicated storage spanning mul-tiple cloud services," in ACM Symposium on Operating Systems Princi-ples, 2013, pp. 292–308.
[12] O. Goldreich, "Towards a theory of software protection and simulation by oblivious rams," in ACM symposium on Theory of computing. ACM, 1987, pp. 182–194.
[13] R. Ostrovsky, "Efficient computation on oblivious rams," in Proceedingsof ACM STOC, 1990, pp. 514–523.
[14] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," Jmynal of the ACM (JACM), vol. 43, no. 3, pp. 431–473, 1996.
[15] E. Kushilevitz, S. Lu, and R. Ostrovsky, "On the (in) security of hash-based oblivious ram and a new balancing scheme," in ACM-SIAM SODA, 2012, pp. 143–156.
[16] P. Williams, R. Sion, and B. Carbunar, "Building castles out of mud: practical access pattern privacy and correctness on untrusted storage," in Proceedings of the 15th ACM conference on Computer and commu-nications security, 2008, pp. 139–148.
[17] P. Williams and R. Sion, "Single round access privacy on outsmyced storage," in ACM CCS, 2012, pp. 293–304.
[18] J. R. Lorch, B. Parno, J. Mickens, M. Raykova, and J. Schiffman, "Shroud: ensuring private access to large-scale data in the data centers," in USENIX FAST, 2013, pp. 199–213.