Do You Know Where Your Cloud Files Are?

Karyn Benson Rafael Dowsley Hovav Shacham[*]

Department of Computer Science and Engineering
University of California, San Diego
La Jolla, California, USA
{kbenson,rdowsley,hovav}@cs.ucsd.edu

ABSTRACT

Clients of storage-as-a-service systems such as Amazon's S3 want to be sure that the files they have entrusted to the cloud are available now and will be available in the future.

Using protocols from previous work on proofs of retrievability and on provable data possession, clients can verify that their files are available now. But these protocols do not guarantee that the files are replicated onto multiple drives or multiple datacenters. Such tests are crucial if cloud storage is to provide resilience to natural disasters and power outages as well as improving the network latency to different parts of the world.

In this paper, we study the problem of verifying that a cloud storage provider replicates the data in diverse geolocations. We provide a theoretical framework for verifying this property. Our model accurately determines which Amazon CloudFront location serves content for Planetlab nodes across the continental US.

Our work is complementary to the recent paper of Bowers et al., which uses different techniques to verify that files are replicated across multiple drives in a single datacenter.

Categories and Subject Descriptors

H.3.2 [**Information Storage**]; D.4.6 [**Security and Protection**]; E.5 [**Files**]: Backup/Recovery

General Terms

Security, Measurement, Experimentation

Keywords

Cloud computing, Geolocation

1. INTRODUCTION

The cloud is quickly becoming a popular medium for storing and computing data. As data is transitioned off traditional servers and into the cloud, users give up direct control of their data in exchange for faster on-demand resources and shared administrative costs. Inherent in cloud computing is a trust and dependency on the cloud service provider. Recently, customers of the Amazon's EC2 cloud service lost data permanently [2] and Hotmail users were upset when it appeared as though their accounts had been wiped [13]. Consequently, it is important that a client be able to verify that the services paid for are the services being provided. It is conceivable that to cut costs a cloud provider may claim one level of service but actually provide an inferior level.

One approach to such verification is to rely on contracts or service-level agreements. But a contractual approach may not detect misbehavior—whether malicious or accidental—until it is too late. It is better to augment such an approach with mechanisms by which clients can themselves verify that the properties they rely on actually hold.

Many aspects of cloud computing can be verified by the end user (speed, number of nodes). In one innovative example, Zhang et al. [18] build on architectural side channels identified by Ristenpart et al. [16] to allow users of infrastructure-as-a-service systems to verify that their virtual machines are not coresident with other users' VMs on a physical machine. For users of storage-as-a-service systems, much work (discussed below) attempts to give users assurance that their files are being stored.

Knowing that one's files are being stored is not always enough. What if they are stored on a single hard disk, and that hard disk fails? Or in a single datacenter that later experiences an outage? What is needed is a proof not just that a file is being stored, but that it is being replicated, onto multiple disks or in multiple geographic locations.

Traditionally, backups are kept in a different location than the main server; resulting in the ability to restore service in the face of power outages or natural disasters. As companies expand globally, replicating data in diverse geolocations to reduces bandwidth consumption and improves access speed. In fact, Amazon SC3 boasts the ability to store data redundantly "on multiple devices across multiple facilities in an Amazon S3 Region [12]."

Accessing the data or executing a traditional Proof of Retrievability only proves that the provider has the data not how many times it is in the cloud. Even if we had a protocol that allowed the user to check that the cloud provider has multiple copies of the data stored, we would still need

[*]Supported by the MURI program under AFOSR Grant No. FA9550-08-1-0352.

to deal with the hard task of assuring that these copies are stored in different geographic locations.

Besides ensuring proper data replication from a contractual standpoint, verifying that data exists in multiple locations has other interesting applications. For instance, a user may not be interested in the precise location their data was replicated but may require for regulatory compliance that their data is stored in multiple locations. Companies purchasing web advertisements may wish to verify that their ad content is displayed to a geographically diverse demographic.

In this work we initiate the study of proofs of geographic replication. The theoretical aspects of the system are explored in Section 3. We address some technical hurdles of implementing the system using Planetlab and build a model to predict geolocation in Section 4. Finally, in Section 5, we successfully use the model to verify the geolocation of data in Amazon's cloud. Our work is complementary to the recent paper of Bowers et al. [3], discussed below, which gives proofs of replication among multiple drives in a single datacenter.

2. RELATED WORK

2.1 Proofs of Retrievability

Proofs of Retrievability (PoR) [9, 17, 5] and Proofs of Data Possession [1] are challenge-response protocols that allow a client to verify efficiently that a file is really stored in a server and that the integrity of the file is preserved. The naive solution to this verification problem is to transmit the whole file from the server to the client, but this is an impracticable approach for large files (which is why users typically store data in third-party servers) since the communication and I/O accessing costs are too expensive. Therefore the PoR schemes try to minimize the communication complexity, the I/O accessing costs and also the storage space required in the server and in the client.

In the particular case of computational unbounded-use PoR (i.e., PoR schemes where a client can run an unlimited number of audit-protocol interactions), the basic idea is to add an authentication tag to each file block [5]. Then the server should store these authentication tags together with each file block and the client should only store the key used to generate the authentication tags in order to run the verification protocol. To verify the response to an arbitrary challenge the client takes the returned blocks, computes the respective authentication tags and checks that they agree with the authentication tags returned by the server. The authentication tags can be, for instance, the result of applying a message authentication code (MAC) to the file block.

2.2 Proof Data was Replicated across Disks

Bowers et al. [3] recently developed a protocol that allows a user to verify that his data is replicated on multiple disks that are located in a single geolocation. The idea behind their scheme is the time that a server needs to answer a request to a group of file blocks depends on the number of drives used to store the file.

More specifically, the adversary considered is a rational one and it is assumed that the client and the server agree upon the exact placement of the files blocks on the distinct drives. The protocol relies on the seek time necessary to locate a random block on a drive (which is the dominant factor in the time taken to access a random small file block. For the purpose of reducing the variance of the sample, several blocks are requested per drive). For the initial step of the verification protocol, the client challenges the server with a query consisting of one random block per drive, according to the agreed-upon layout. For the following steps, the challenge blocks, also one per drive, are dependent on the contents of all the blocks read in the previous step (therefore the adversary cannot schedule the block access in a convenient way to improve performance). Note that a honest server can read all the blocks in parallel in each step, while dishonest servers cannot. Thus, measuring the time spent on the request, the client can test if the cloud provider is storing the files with the agreed level of redundancy. Of course in practice there are some complications related to the variance in network latency and disk access time, but the authors were able to overcome these difficulties and set up an experimental system. Particularly, using random blocks of 64KB in the request, the authors noticed that the disk access times are similar for modern enterprise class drives of different manufacturers. Therefore, imposing a time constraint in the protocol, the authors were able to develop a scheme to ensure that the data is replicated on multiple disks.

The protocol provided by Bowers et al. in [3] provides resilience to drive crashes and failures but does not readily extend to the multiple geolocation setting. For all steps except the first one, the challenge blocks are dependent on the contents of all the blocks read in the previous step. Hence all disks are required to be in the same place for the protocol to execute correctly within the time constraints.

Our proposed solution also uses the idea of imposing a time constraint, but in a different way and with a different goal. We put together a time constraint and some assumptions about the cloud provider in order to obtain a scheme that allows an user to check that the cloud provider is keeping at least one copy of the file in each agreed geolocation.

2.3 Geolocation

There are many methods to find the estimated geolocation of a host. Domain registration records, accessible through whois, provide the physical address of the registrant for an IP address; however, domain registration records have the potential to be both outdated or related to an administrative site instead of the actual location. Sites such as `ip2location.com` and `ipligence.com` have large, for-pay databases that provide an estimate of where an IP address is located. Although, these services seemed to be more accurate than whois, an adversary can dynamically reassign IP address - which would probably not be reflected in a static database. Traceroute discovers the path packets traverse using ICMP or TCP. Unfortunately, an intermediate router may drop these packets leaving only a partial route.

The real problem with traceroute and whois services is that they use an IP address as input. Colluding storage nodes have the ability to send packets with the same IP address. As a result, a distant machine could respond to storage challenges even though the IP address appears to be close.

Several research projects have studied the problem of geolocating a machine using latency. Katz-Bassett et al. [10] provided a comprehensive study of geolocation techniques.

- *GeoPing* works under the assumption that if two hosts are near each other then their delays will be similar to other landmarks. After pinging a target from all landmarks, the result is a display vector which reveals how close a target is to the landmarks.

- Introduced by Gueye et al. [7], *Constraint-Based Geolocation* uses a triangulation technique to determine the positioning of a target, which is possibly in between the landmarks.
- *Shortest Ping* The authors of [10] ping the target from all the landmarks. The target is then closest to the landmark with the shortest ping.
- *Speed of Internet* By using many planet lab measurements, in [10] the authors determine that the end to end time of most packets has an upper bound of approximately $\frac{4}{9}c$ where c is the speed of light.
- *Topology Based* The primary result of [10] is that geolocation is more accurate when topology is considered. Using traceroute, topology and per-hop latencies are inferred. A constraint-based optimization attempts to geolocate the target and all intermediary routers.

The work by Katz-Bassett is by no means the only work attempting to discover the location of a host based on measurements. Francis et al. [6] find the nearest host in terms of low latency or high bandwidth. Systems such as Vivaldi [4] use a synthetic network coordinate systems to predict the latency between the hosts. More recently, Laki et al. [11] used a path-latency model to separate out different types of network delay. They conclude that accurately estimating the propagation delay can improve the accuracy of distance estimation.

3. MODELING THE CLOUD AND A THEORETICAL SOLUTION

We present a solution for the case in which a customer has contracted a service to store multiple copies of his file in different geolocations and wants to verify the service level. Note that keeping backups in different geolocations is a good idea in order to prevent against unavailability caused by factors such as natural disaster and power outages, and also to improve the network latency to access the data in different parts of the world. We should also stress that these advantages are lost in a great extension if the copies are stored in the same physical place or in places that are very close.

Let C be a cloud provider that has n datacenters, denoted by s_i for $1 \leq i \leq n$. Suppose that a customer made a contract with C to keep one copy of his file in each of the datacenters $s_{i_1}, s_{i_2}, \ldots, s_{i_k}$. We want to build a protocol that tests if the cloud provider is really keeping one copy of the file in each of the specified datacenters.

In order to construct the protocol, we make some assumptions about the cloud provider C. We assume that the adversary will try to cut costs, but that he is not malicious. We state below the assumptions that we made about C. These assumptions are in line with the principle that the adversary is only trying to reduce the operational costs.

ASSUMPTION 1. *The locations of all data-centers of the cloud provider C are well-known and all the data is stored in these places.*

ASSUMPTION 2. *The cloud provider does not have any exclusive Internet connection between the datacenters.*

Although the adversary is economically rational, it may still deviate from the protocol when it believes it is being audited for location redundancy. A storage verification protocol should not make excessive assumptions about the datacenter and its server. Additionally, audits should be not tip the cloud services provider as to what data is being audited. The auditor is able to make assumptions about the machines it uses in the audit.

ASSUMPTION 3. *For each datacenter s_i, we have access to a machine t_i that is located comparatively very close to s_i (considering network latency as the distance function), when compared to its distance to the other data centers and to the distances between s_i and the other datacenters.*

Note that in some cloud providers such as Amazon, it is actually possible to execute a virtual machine inside the datacenter and so we have the ideal machine t_i for our purposes.

We can estimate the network latency (in this work we consider one-way latency when we do not specify otherwise) between each pair of datacenters by using clients positioned very close to the location of the datacenters. This can be done due to the fact that the latencies have a big geographical correlation [3]. So, we make the following assumption about the resources that we have to perform the test.

For $a, b \in \{s_1, \ldots, s_n, t_1, \ldots, t_n\}$, let $LB(\alpha, a, b)$ be the value such that the network latency from a to b is greater than $LB(\alpha, a, b)$ with probability at least α (normally the interesting values of α are the ones that are very close to 1). Now we compute

$$MLT(i) = \min_{1 \leq j \leq n : j \neq i} (LB(\alpha, s_i, s_j) + LB(\alpha, s_j, t_i))$$

Note that for any $j \neq i$, we have that the network latency to send something from s_i to s_j and from there to t_i is equal or greater than $MLT(i)$ with probability at least α^2.

Let $\widehat{LB}(\alpha, t_i, s_i)$ and $\widehat{UB}(\alpha, t_i, s_i)$ be the values such that, with probability at least α, the network latency from t_i to s_i is greater than $\widehat{LB}(\alpha, t_i, s_i)$ and smaller than $\widehat{UB}(\alpha, t_i, s_i)$ and such that the value of $\Delta(\alpha, t_i, s_i) = \widehat{UB}(\alpha, t_i, s_i) - \widehat{LB}(\alpha, t_i, s_i)$ is minimized.

We want to construct a protocol to test if a copy of our file is present in s_i. Let T_i be an upper bound on the execution time of some auditing protocol. If the cloud provider is honest, the client should accepted the protocol execution as valid even if the elapsed time is $T_i + 2\widehat{UB}(\alpha, t_i, s_i)$, since this is the time needed to receive the answer in the worst case scenario. On the other hand, the protocol execution should not be accepted as valid if the elapsed time is greater than $MLT(i) + \widehat{LB}(\alpha, t_i, s_i)$, since in this case a dishonest provider can use data present in another datacenter. Therefore we should have

$$\begin{aligned} T_i &\leq MLT(i) + \widehat{LB}(\alpha, t_i, s_i) - 2\widehat{UB}(\alpha, t_i, s_i) \\ &= MLT(i) - \widehat{UB}(\alpha, t_i, s_i) - \Delta(\alpha, t_i, s_i) \end{aligned}$$

Hence, with probability at least α^3, any verification protocol in which the server has time limit T_i to respond, the cloud provider can only use data that is present in s_i.[1]

[1]The term α^3 is the level of confidence that in the triangulation possibly used by a dishonest provider all the latencies will be within the used bounds.

3.1 Using Proofs of Retrievability in the Cloud Model

For typical values of T_i we do not have enough time in order to execute the known Proofs of Retrievability protocols [9, 17, 5]. But as long as T_i is big enough to execute a challenge to at least one random small block, we can still use the basic idea behind computational unbounded-use PoR (i.e., authentication tags that are associated with each file block, stored in the cloud and used in the verification by the user that only keeps the authentication key) in order to obtain a verification protocol to test server s_i.

In each protocol execution the client sends a challenge with as much random blocks as it possible to access in time T_i. If the answer is not received in time $T_i + 2\widehat{UB}(\alpha, t_i, s_i)$, the client considers this an invalid execution. In the case that a fraction substantially larger than $1 - \alpha^2$ of the executions are invalid, the client knows that the server is dishonest. If the returned pair (file block, authentication tag) is not correct for some challenged block, then the client knows that his file is not stored correctly in s_i. Therefore if the server is only storing a fraction $f < 1$ of the total file size, then the probability that he succeeds in the tests decreases exponentially with the number of challenged blocks. To increase the confidence level, the client can keep running executions of the protocol at random times and at each instant he can use the t more recent (valid) challenged blocks to decide if the server is storing the file correctly or not.

Another approach that may be possible, depending on the specific value T_i and on the size of file, is to read a sequential part of the file in each execution. This approach exploits the fact that the seek time of the disks are relatively high, but they can normally read data at sequential positions very fast. But using such approach, the probability of detecting that the server is not storing your file correctly only decays linearly in the amount of data read in the execution. In addition, this approach also significantly increases the amount of data send between the server and the client.

3.2 Theoretical Limitations

Note that if some group of datacenters are very close to each other, then T_i will be very small and will not allow us to test even one random small block. In this case we cannot check if a copy of the file is present in one specific datacenter of this group or not. But we still may be able to distinguish a copy present in this group of data centers (without knowing in which specific one) from a copy of the file present in other, distant, datacenters. I.e., let $S = \{s_1, s_2, \ldots, s_n\}$ and suppose that there is a group of data centers $\{s_{i_1}, s_{i_2}, \ldots, s_{i_k}\} \subset S$ that are close to each other, but far away from the remaining datacenters $S \setminus \{s_{i_1}, s_{i_2}, \ldots, s_{i_k}\}$. Then we can use similar ideas to test if there is at least one copy of our file in the group of datacenters $\{s_{i_1}, s_{i_2}, \ldots, s_{i_k}\}$.

Hence, we can group the datacenters in accordance with their geographical region and verify that there is at least one copy of the file in each geographical region. We should stress again that in many situations this is exactly what the customer wants, one replica of his file in each region, in order to prevent against problems such as power outage and natural disaster and also to improve the latency experienced by users in all parts of the world.

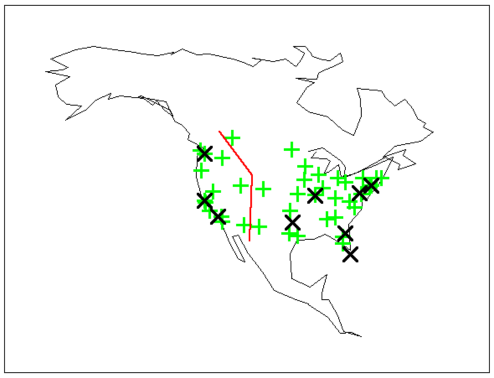


Figure 1: 40 Planetlab Nodes and 2 Amazon S3 Server Farms

4. BUILDING A PRACTICAL VERIFICATION SYSTEM

The verification protocol requires that the test node be able to determine whether the response to a challenge came from the datacenter being tested, or another datacenter. Specifically, knowledge of the datacenter locations and the latencies between them is needed. In this section we show that determining this information is feasible on today's Internet.

We also address the technical problems that arise in creating a verification system. We set out to answer:

- Which model gives the best prediction of geolocation?
- How far apart do datacenters need to be to accurately discriminate the location of data?
- What are the characteristics of a node that accurately decides geolocation?
- Which passive distance measurement is best in predicting geolocation?

4.1 Determining the Origin of Data from Latency

Individual hosts can measure the latency between themselves and other hosts on the Internet. Extracting the origin of the data requires building a model to estimate the distance between the hosts.

Passive Distance Measurements

There are several ways of calculating the actual distance between two points corresponding to hosts on the Internet.

- Haversine Distance: This is the distance between two points on a sphere. There are models that take into account the exact shape of the earth; but the haversine distance seems to be fairly accurate in calculating the shortest distance between cities in North America.
- Driving Distance: We used the Google Map API to calculate the distance between the cities. Fiber optic cable is typically laid between major cites, similar to the Interstate Highways. Moreover, routing decisions for both roads and packets may be based on geologic

structures. For example, in Figure 1 the red line separates nodes based on which Amazon S3 Server Farm was used (east or west coast). This line is also overlays the Rocky Mountains. Several sites just east of the Rocky Mountains use an east coast server instead of a west coast server they are closer to the west coast location.

- Topology Based Distance: This measurement yielded the best results in [10]. However, this method requires using traceroute to determine the per-hop latency between each router. Our goal is to have a simple distance calculation that assists in geolocating an IP address without probing that would tip off the cloud storage provider of an audit.

In this paper, the haversine distance is used except when noted.

Latency on the Internet

We used Planetlab to get a sense of latency on the Internet. We measured the time between sending a SYN packet and receiving a SYN-ACK packet of a TCP connection half of which is considered the latency. Note that this includes the latency between the nodes as well as the time to set up a TCP connection on the remote server.

Geolocation Predictive Models

Having collected latency measurements from test nodes to known locations the following methods could be used to determine the distance away subsequently downloaded content originated from:

- Previous work [10] indicated that $\frac{4}{9}$*(speed of light) is a good upper bound on Internet latency.
- The best fit line for a sampling of nodes. Given a collection of (distance, latency) measurements to known hosts we can use linear regression to find the best fit line that models the relationship. Using reliable nodes, we estimate latency (in ms) to be $1.202 \times 10^{-5}d + 0.0007701$ where d is the distance in km. This differed from $\frac{4}{9}$ method; reasons for this discrepancy are most likely from the way distance was measured, the time to set up state for a TCP connection, and the original approximation was for fiber optic cables (the Planetlab hosts may not be connected directly to fiber).
- Given measurements for a particular host, we can calculate the equation of the linear regression of these points. We call this the site (or node) expected method. This method should be useful for sites located far from the core Internet routers. For example, traceroute revealed that packets sent from the University of Nevada, Reno sent to Seattle first travel 100 miles south to Sacramento before being routed 700 miles north to Seattle.

Feasibility

We expect latency and distance to have a positive linear relationship. Figure 1 shows 40 Planetlab nodes dispersed across North America from which we measured the latency when downloading an image suspected to be in Amazon's S3. The requests were served by two class C networks: one for east coast Planetlab nodes and another for west coast Planetlab nodes. Amazon is based in Seattle, WA, but has several secondary locations including Palo Alto, CA. If datacenter geolocation is feasible then we expect a positive linear relationship between latency and the distance from the host to the location of the actual datacenter. The other location should not have a positive linear relationship between latency and the distance from the host to the alternate location. From the graph in Figure 2, it was clear that the data originated from Seattle and not Northern California.

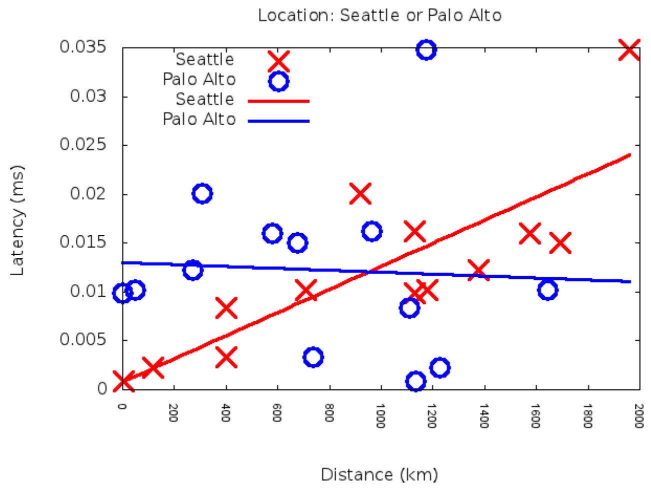


Figure 2: Palo Alto or Seattle?

4.2 Best Distance and Predictive Models

One way a content provider routes requests is through DNS redirection [15]. Using public DNS servers in San Jose, California and New Jersey, it was possible to force the Planetlab nodes to download the image from the site of our choosing. For each of our Planetlab nodes, we downloaded an image from Amazon's S3 from both Seattle, WA and Ashburn, VA 75 times. All of these points were used as training data to calculate the best fit and the node expected functions.

Our first experiment had each Planetlab download content from 40 universities in North America. Under the assumption that these institutions host their own web content, the haversine distance was calculated as the truth value. The predictive models were used to calculate an estimate for the distance away.

The average absolute value of the error in the distance calculation is graphed in Figure 3a. Two nodes, Cornell and Iowa State had a significantly larger average difference than the other sites (more than 100,000 km). One reason for the large error may be the quality of the node's Internet connection. The node at Cornell was named planetlab4-dsl.cs.cornell.edu, which seems to indicate that it uses a slower DSL connection instead of Ethernet.

Excluding these two sites greatly improved the predictive power of the best fit method, as seen in Figure 3b. Still, two sites (University of Arizona and University of Colorado, Boulder) performed poorly in predicting the site; especially the $\frac{4}{9}$ and Site Expected methods. Poor performance in the Site Expected method indicates that these sites did not produce consistent data over the Amazon S3 and University Datasets. We also excluded data from the two additional

Dist. Meth	Model	Avg. Error	Std. Dev.
Haversine	$\frac{4}{9}$	1526.92	1012.672
Haversine	Best Fit	441.637	459.592
Haversine	Site Expected	531.792	526.440
Driving	Best Fit	514.217	542.741
Driving	Site Expected	629.340	608.167

Table 1: Average Error for Different Predictive Models

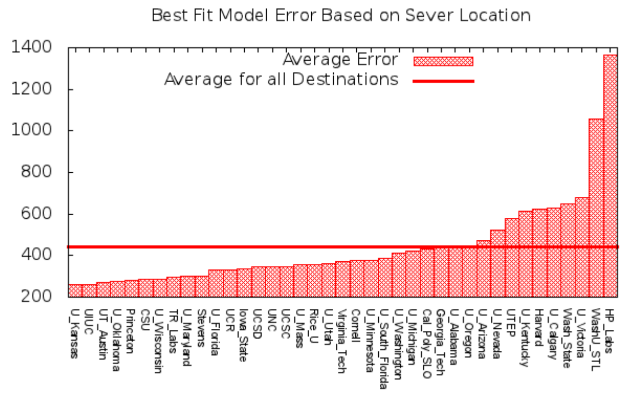


Figure 4: Distance Error by Server

nodes (Washington University in St. Louis and the University of Victoria in British Columbia), after repeating the pruning and seeing that these nodes performed poorly on all three measurements. The result is seen in Figure 3c[2].

From these graphs, the $\frac{4}{9}$ method is clearly the worse predictive model. The Site Expected model outperforms the Best Fit model when there are unreliable nodes. However, once the unreliable nodes are eliminated the Best Fit model performs the best in terms of average error (see Table 1). It is not surprising that the Best Fit and Site Expected models outperform the $\frac{4}{9}$ model as the $\frac{4}{9}$ the speed of light is a quick upper bound approximation.

We were surprised to see that the Best Fit model outperformed the Site Expected at 29 of the 34 reliable sites. This indicates that taking into account local routing information does not improve the predictive capability of the model. A plausible explanation for this is that there are delays that are incurred near both the server and the client. Our training data only used two sites to build the models - both Amazon servers. The servers at the universities are heterogeneous so the Site Expected model does not perform well on machines with different configurations. We can see in Figure 4 that the average error differs greatly between sites.

The best predictive model, for destinations of unknown configuration, is to use measurements of latency and distance from diverse but reliable hosts.

Errors within one standard deviation, which is approximately 900 km (441km ± 459 km), seem reasonable to tell the distance between many cities. The graph in Figure 5 shows the percentage of measurements that had an error

[2]Note these graphs have different scales. Specifically, we use a logarithmic scale in Figure 3a and a linear scale in the other graphs

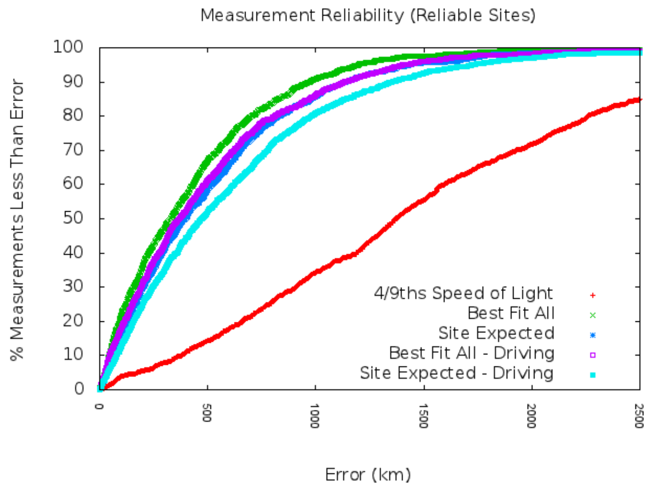


Figure 5: Cumulative Distance vs. Error (Reliable Sites)

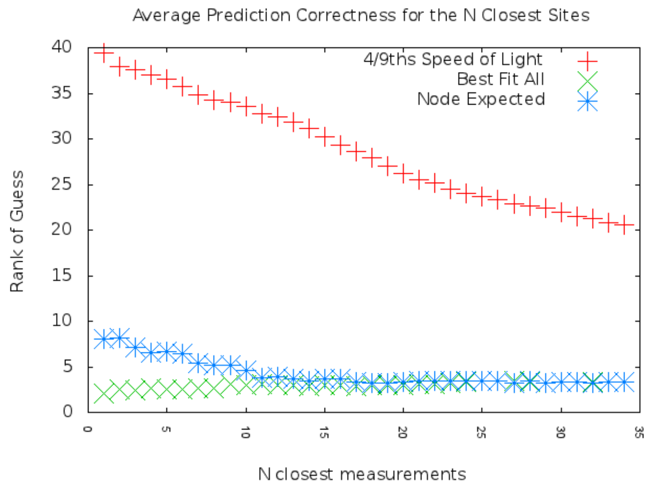


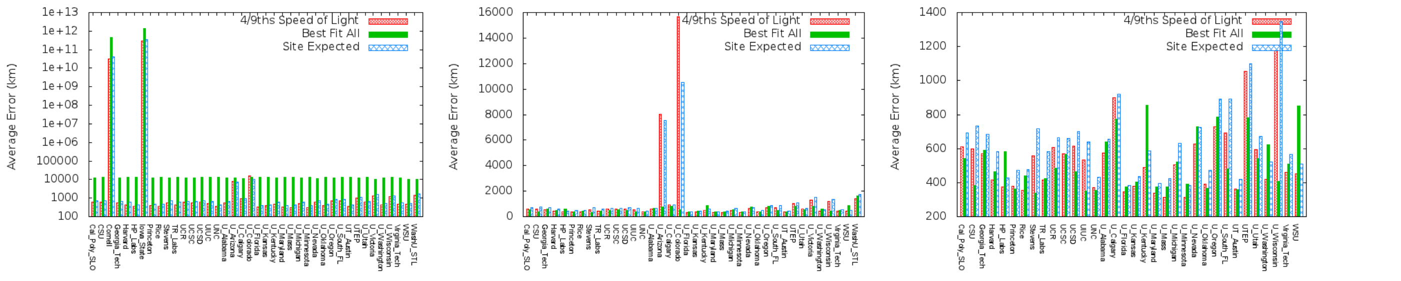
Figure 6: Average Ranking for the N closest Sites

less than a given distance. For the Best Fit model, we can get 90% confidence at approximately 975 km and 95% confidence at approximately 1200km. Additionally, driving distance did not improve the prediction.

4.3 Effect of Number of Nodes on Accuracy

The previous experiments showed that it was possible to coarsely estimate the distance a server is away. An error of 900-1200km is reasonable to tell the difference between most US cities, but is by no means precise. Additional measurements should help decrease the error, by means of triangulation. This will help us find the precise location of datacenters.

Using the same data from the previous experiment, we had each Planetlab node rank the universities from most likely (rank 1) host to least likely (rank 40) in addition to calculating the distance. We consider the average ranking of the n closest nodes (and average this over all 40 sites) in Figure 6.



(a) All University Data (b) Error < 100000 km (c) Reliable Nodes

Figure 3: Determining Node Reliability

Using more measurements clearly helps the certainty and accuracy of the $\frac{4}{9}$ and Site Expected models. For the Best Fit model the number of measurements doesn't seem to affect the predictive capability of the model (although the best performance was with 1 or 5 nodes contributing to the estimated location).

5. VERIFYING STORAGE LOCATIONS

Universities and their websites are only in one location. We are interested in developing an algorithm to figure out the origin of data when there is more than one location. To do this, we created an account on Amazon's CloudFront and uploaded a file. We then made a HTTP request for our file from the reliable Planetlab nodes and recorded the latency. The image was fetched three times from each node, almost every time the server sending the image had a different IP address.

5.1 Finding Datacenters

Using Assumption 1, we know the location of Amazon's CloudFront datacenters: Ashburn, Dallas, Jacksonville, Los Angeles, Miami, New York, Newark, Palo Alto, Seattle and St. Louis. It is unreasonable to expect to be able to tell the difference between New York and Newark as they are 14.2 km away, so they were considered one node - called New York. Each Planetlab node ranked all of the sites from most likely (rank 1) to least likely (rank 9). Two variations of the ranking were tested: considering only the n highest ranks, only ranking the locations who were estimated to be within a certain distance.

The hierarchical clustering algorithm given in Algorithm 1 was used to cluster the results.

$\mathsf{euclidian-distance}$ is the standard algorithm. The function $\mathsf{weighted\text{-}ranks}$ aims to average the ranks of the two most similar clusters. However, if cluster$_1$ was comprised of n samples and cluster$_2$ was comprised of m samples we want the resulting average to reflect any potential imbalance in cluster size. We say cluster$_1$ has weight n and cluster$_2$ has weight m, and calculate the new ranking of a datacenter to be $\frac{n}{n+m}$(cluster$_1$'s ranking) + $\frac{m}{n+m}$(cluster$_2$'s ranking).

In general, only the top ranked city is used- which is obtainable by setting the threshold to $0+\epsilon$. As a consequence, we may obtain a couple clusters that share the same top rank but differ in the second or third rank. An alternative is to use only the sites that are suspected to be within a given radius (say 500 km). Some measurements will be classified as unknown if the suspected distance is greater than this threshold. This helps eliminate packets that have an abnormal delay.

Algorithm 1 Hierarchical Clustering Algorithm

```
Input: rankings, threshold
  clusters ← rankings
  weights ← [1,···,1]
  min_sim ← −∞
  first ← True
  while min_sim ≤ threshold do
    if !first then
      add to rankings weighted-ranks(clusters[join_1], weights[join_1], clusters[join_2], weights[join_2])
      remove rankings[join_1] and rankings[join_2]
      add to weights[join_1]+weights[join_2]
      remove weights[join_1] and weights[join_2]
    end if
    first ← False
    min_sim ← ∞
    for $i = 0 \rightarrow |\text{clusters}|$ do
      for $j = 0 \rightarrow i-1$ do
        sim ← euclidian-distance(clusters[$i$], clusters[$j$])
        if sim < min_sim then
          min_sim ← sim
          join_1 ← $i$ ; join_2 ← $j$
        end if
      end for
    end for
  end while
```

To evaluate the system, domain name resolution was used to find the common names of the Amazon servers from which the geolocation could be deduced (server-216-137-47-169.mia3.cloudfront.net is most likely in Miami). Several parameters were tested: counting the highest ranked guesses of geolocation, using only measurements that put the datacenter close to the Planetlab node, and a combination of the these parameters. The F-1 score[3] was then calculated for each site.

The results of the F-1 metric seem to indicate that selecting only the top choice outperforms selecting all of the

[3]The F-1 score is the harmonic mean of precision and recall. A F-1 score of 1.0 is the best while a score of 0 is the worst.

Test	Ashburn	Dallas	JAX	LAX	Miami	NYC	Palo Alto	STL	Seattle
Top Rank	0.88	0.87	0.27	1.0	0.0	1.0	0.91	0.65	0.92
Top 2 Ranks	0.4	0.87	0.22	0.69	0.73	0.73	0.29	0.48	0.92
Top 3 Ranks	0.32	0.25	0.16	0.48	0.24	0.32	0.27	0.36	0.22
< 250 km (All Ranked)	0.31	0.0	0.0	0.57	0.0	0.84	1.0	0.33	0.31
< 500 km (All Ranked)	0.81	0.7	0.0	0.74	0.71	0.92	0.71	0.70	0.84
< 750 km (All Ranked)	0.56	0.64	0.22	0.61	0.71	0.67	0.48	0.55	1.0
< 1000 km (All Ranked)	0.49	0.74	0.42	0.77	0.94	0.56	0.47	0.48	0.97
< 250 km (Top Rank)	0.31	0.0	0.0	0.57	0.0	0.84	1.0	0.33	0.31
< 500 km (Top Rank)	1.0	0.7	0.0	0.82	0.0	1.0	0.91	0.70	0.84
< 750 km (Top Rank)	0.88	0.7	0.27	0.82	0.0	1.0	0.91	0.55	1.0
< 1000 km (Top Rank)	0.88	0.87	0.27	1.0	0.0	1.0	0.91	0.55	1.0

Table 2: F-1 Score For Various Parameters

ranked scores. Specifically, the F-1 score decreased between thresholds of 500 km and 750 km for two of the cities; however, the F-1 score increased for two other cities. This means above 500 km there are tradeoffs between selecting more data points for increased certainty and accuracy.

The accuracy is determined by how reliable the nodes are - which we measured in the University experiment. As more sites concur that the data is in a location the more certain we become that our data is stored there. One potential problem with this dataset is that the number of nodes going to each datacenter is not evenly distributed. There are only two sites that go to Jacksonville, FL. At the high end there are five sites that use St. Louis, MO.

In verifying storage, we are more interested in accuracy than certainty. So it is reasonable to conclude that restricting predictions to only use measurements under 500km is best. A slight modifications to the F-1 score weights precision higher than recall. This could be a more appropriate measurement to reflect our priorities.

A threshold of 500km is reasonable for this data set. Almost all of the Planetlab nodes are within 500km of the closest Amazon datacenter. We notice that St. Louis consistently performed poorly in comparison to other sites with the different parameters. This may be attributed to routing abnormalities in the city of St. Louis itself; see Figure 4 where the University of Washington in St. Louis had the second worst performance. Or this could be attributed to St. Louis' central location. Close proximity to other sites requires that the threshold is lower to keep the same level of accuracy.

Another way of looking at this data is on a Map 7. This graph is of the top choice when restricted to predictions of less than 500 km. In this graph, a correct assignment of an IP address to city is designated by a green circle; similarly, an incorrect assignment is designated by a red circle. The map is comprised of 72 measurements. Missing from this map are 30 measurements because the estimated distance away was greater than 500 km: nine that were served from Miami, six from Dallas and Jacksonville, and three each from Los Angeles, Seattle and St. Louis.

Of the incorrect selections, nine that were actually routed from Miami, FL were estimated to be in Jacksonville, FL. Jacksonville and Miami are approximately 500 km apart. This seems to indicate that 500 km is too close for the datacenters to be to accurately differentiate where the data came from using this method. The only other incorrect selection was a single IP address located in St. Louis, MO but was

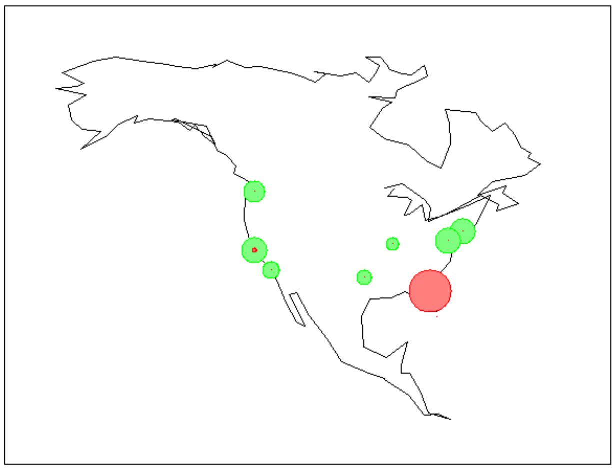


Figure 7: Top Choice of Amazon CloudFront Datacenter

estimated to be in Palo Alto, CA. Although the cities are a fair distance apart, it is unsurprising that we have errors at the central location of St. Louis.

This experiment showed, that with the proper parameters we are able to predict with a high degree of accuracy where data originates.

5.2 Finding Locations Without Assumption 1

It seems reasonable to expect that a contractual agreement between a cloud service provider and a user would divulge the locations of the datacenters. However, our experience in creating accounts with several leading cloud storage providers indicated that this information was not always readily available. Our method of locating datacenters extends to the situation where the locations are not well known but with a performance hit.

The main technical difficulty is in selecting the target locations for where the data may be located. We ran our hierarchical clustering algorithm using the latitude and longitude of the 30 largest cites [8] in the US as input. Increasing the number of cities from nine to 30 also decreased the distances between potential server locations. To compensate, the threshold for which to stop the hierarchical clustering can be increased. Three of the nine sites (counting Newark and New York as one site): Ashburn, Miami and St. Louis

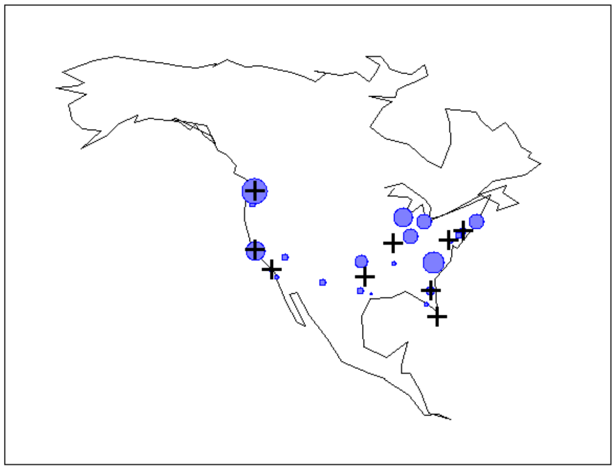


Figure 8: Top Choice of Amazon CloudFront Datacenter with Unknown Locations

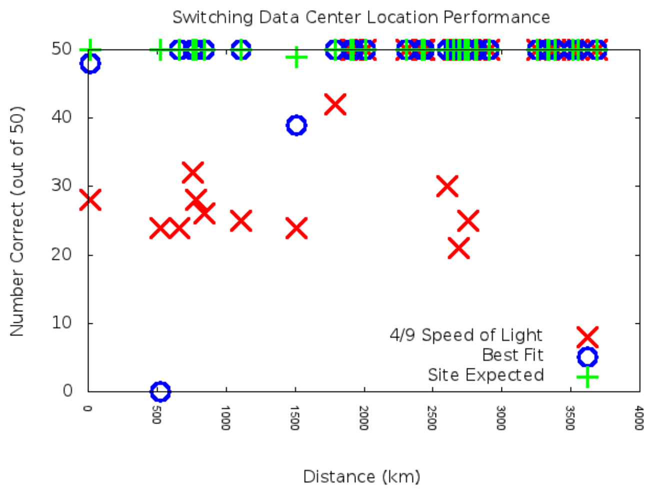


Figure 9: Predicting Datacenter after Switching DNS Server

were not on the list of the 30 largest cities in the US. Consequently, it is impossible to find an exact match on these sites.

Figure 8 maps the predicted geolocations using the parameters of the top site within measurements less than 500 km. The actual locations are marked with an X. The radius of a circle around a city is proportional to the number of IP addresses estimated to be at that city.

The results show that removing Assumption 1 gives a coarse idea of where the datacenters are located.

On the west coast, the datacenters in Seattle and San Francisco/Palo Alto were accurately discovered. There were only two Planetlab sites that used the Los Angeles datacenter and there is an appropriate small cluster of estimates in the southeast corner of the United States.

In the Midwest the Planetlab sites could, with the exception of one measurement, determine that they were being served content from the Midwest. Traffic actually originating St. Louis (not on the top 30 cities) was predicted to be at many of the surrounding cities. Milwaukee, Indianapolis and Detroit were each associated with at least three IP addresses. Despite Dallas being on the top 30 cities list, no Planetlab site connecting with Dallas pick it as the city of the origin. Nearby Oklahoma City and and Austin were popular estimates along with El Paso (900 km away).

The densely populated, east coast was less accurate. New York is situated between Philadelphia and Boston; all of which received votes as the location of the New York and Newark datacenters. When a Planetlab node connected to an Amazon datacenter in Miami, more than half the time it predicted it was in Florida (three IP addresses were in Charlotte, NC). Although Ashburn is located approximately 50 km from Washington DC, this was the estimated geolocation three times; nine times the second closest site Charlotte, NC (600km away).

Although we cannot find the exact location, knowing the approximate location suffices for our application. As discussed in Section 3.2 when storing data in the cloud customers are more concerned about the diversity of the locations than the exact locations.

5.3 Detecting Changes in Location

A requirement of verifiable storage is the ability to detect when the cloud storage provider switches the location from which data is served. In general, testing where the data came from should suffice. However the setting is different in two key ways:

- It should be able to verify that even a single packet came from an unexpected server.
- The expected location is well known, and there is a large amount of historical data that allows for more accurate prediction of the latency between the node and the cloud storage provider.

We did a test to see if the reliable Planetlab nodes could tell the difference between a download from Ashburn, VA or Seattle, WA. The Planetlab nodes first downloaded the image many times from each Ashburn and Seattle.

We wrote a script that randomly set the DNS server and made a HTTP GET request for the image. The Planetlab nodes were randomly assigned DNS server to download the image, and calculated the expected distance, e. It then output a guess of whichever datacenter location was closer to e. The node's success in predicting the origin is shown in Figure 9. As our distance measurement, we use the absolute value in the difference between the distance from the Planetlab node to each S3 server farm. This effectively tells us how much closer the near Planetlab node is than the far Planetlab node. We expect Planetlab nodes with larger distances to perform better than those with smaller distances.

All prediction methods can tell origin of the data when the distance more than 3000 km; with very few exceptions, the prediction is always correct when the distance is more than 1500 km. The Best Fit and Site Expected methods clearly outperform the $\frac{4}{9}$ method. Unlike the previous experiments, the Site Expected method outperforms the Best Fit at a few data points. This can be attributed to the fact that the datacenter machines are most likely homogeneous, and are therefore more predictable. This is consistent with Figure 4 where we saw that server heterogeneity adversely affected the prediction.

This experiment should be repeated over a period of time. The collection was completed in a single evening. This did not account for changes in routing predictability due to the time of day (we expect more congestion and therefore more latency during waking hours than during the middle of the night) or changes to the route (different routing pattern, changes to availability or hardware at any intermediate router).

6. CONCLUSIONS AND FUTURE WORK

In this paper we presented a solution to the timely problem of verifying the geolocation of data in the cloud. We provided a theoretical framework to set up a verification system and suggested ways to extend traditional proofs of Retrievability to the application. We used Planetlab to collect data from diverse locations and concluded that choosing reliable nodes is paramount to the accuracy of our algorithm. Finally, we successfully identified the approximate geolocations of data in Amazon's cloud.

Ideally, we would extend the theoretical framework and experiments to provide an independent service that audited cloud storage providers. In order to optimize this solution we would test additional cloud storage providers besides Amazon. Running the experiments at different times of the day and over a long period of time could provide a more accurate model of the provider's system and its response to stress.

The latency to geolocation experiments in this paper involve fetching small files over the Internet. In order to build a functional end-to-end system several modifications are needed. In the experiments, we do not address packet loss (although we would probably exclude nodes with high loss rate). It is possible that the service provider would purposefully drop audit packets. The relationship between retransmission and latency in this setting is an open problem.

A complete system would leverage PoRs to provide a high level of confidence of storage through selectively querying datacenters to establish geolocation. PoRs are more complex and require additional computation and time. It seems reasonable to assume that the time to read from disk and perform computations would be the same across datacenters; however, the model would need to be adjusted to account for this extra time. Additionally, the ideas from section 3.1 would need to be applied to use traditional PoRs.

The simple hierarchical clustering algorithm grouped similarly ranked IP addresses to find the datacenters. Advanced clustering algorithms such as spectral clustering may yield better results - especially when the locations of the datacenters are unknown.

7. REFERENCES

[1] G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, and D. X. Song. Provable data possession at untrusted stores. In Ning et al. [14], pages 598–609.

[2] H. Blodget. Amazon's cloud crash disaster permanently destroyed many customers' data. `http://www.businessinsider.com/amazon-lost-data-2011-4`, 2011.

[3] K. D. Bowers, M. van Dijk, A. Juels, A. Oprea, and R. L. Rivest. How to tell if your cloud files are vulnerable to drive crashes. Cryptology ePrint Archive, Report 2010/214, 2010. `http://eprint.iacr.org/`.

[4] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '04, pages 15–26, New York, NY, USA, 2004. ACM.

[5] Y. Dodis, S. P. Vadhan, and D. Wichs. Proofs of retrievability via hardness amplification. In O. Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 109–127. Springer, 2009.

[6] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. F. Gryniewicz, and Y. Jin. An architecture for a global internet host distance estimation service. In *IEEE INFOCOM*, pages 210–217, 1999.

[7] B. Gueye, A. Ziviani, M. Crovella, and S. Fdida. Constraint-based geolocation of internet hosts. In *IEEE/ACM Transactions on Networking*, pages 288–293, 2004.

[8] C. in the United States. `http://www.travelmath.com/cities-in/United+States`.

[9] A. Juels and B. S. K. Jr. Pors: proofs of retrievability for large files. In Ning et al. [14], pages 584–597.

[10] E. Katz-Bassett, J. P. John, A. Krishnamurthy, D. Wetherall, T. E. Anderson, and Y. Chawathe. Towards ip geolocation using delay and topology measurements. In J. M. Almeida, V. A. F. Almeida, and P. Barford, editors, *Internet Measurement Conference*, pages 71–84. ACM, 2006.

[11] S. Laki, P. Matray, P. Hága, I. Csabai, and G. Vattay. A model based approach for improving router geolocation. *Computer Networks*, 54(9):1490–1501, 2010.

[12] A. W. S. LLC. Amazon simple storage service (amazon s3). `http://aws.amazon.com/s3/#protecting`, 2010.

[13] D. Murphy. Hotmail users report blank inboxes. `http://www.pcmag.com/article2/0,2817,2374949,00.asp`, 1 2011.

[14] P. Ning, S. D. C. di Vimercati, and P. F. Syverson, editors. *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*. ACM, 2007.

[15] M. Rabinovich. Issues in web content replication. *IEEE Data Eng. Bull.*, 21(4):21–29, 1998.

[16] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In E. Al-Shaer, S. Jha, and A. D. Keromytis, editors, *ACM Conference on Computer and Communications Security*, pages 199–212. ACM, 2009.

[17] H. Shacham and B. Waters. Compact proofs of retrievability. In J. Pieprzyk, editor, *ASIACRYPT*, volume 5350 of *Lecture Notes in Computer Science*, pages 90–107. Springer, 2008.

[18] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter. HomeAlone: Co-residency detection in the cloud via side-channel analysis. In *Proceedings of IEEE Security and Privacy ("Oakland") 2011*. IEEE Computer Society, May 2011.