Implementation of Cloud-RAID: A Secure and Reliable Storage above the Clouds

Maxim Schnjakin and Christoph Meinel

Hasso Plattner Institute, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
maxim.schnjakin, meinel@hpi.uni-potsdam.de

Abstract. Cloud Computing as a service-on-demand architecture has grown in importance over the previous few years. One driver of its growth is the ever increasing amount of data which is supposed to outpace the growth of storage capacity. In this way public cloud storage services enable organizations to manage their data with low operational expenses. However, the benefits of cloud computing come along with challenges and open issues such as security, reliability and the risk to become dependent on a provider for its service. In general, a switch of a storage provider is associated with high costs of adapting new APIs and additional charges for inbound and outbound bandwidth and requests. In this paper, we describe the design, architecture and implementation of Cloud-RAID, a system that improves availability, confidentiality and integrity of data stored in the cloud. To achieve this objective, we encrypt user's data and make use of the RAID-technology principle to manage data distribution across cloud storage providers. The data distribution is based on users' expectations regarding providers geographic location, quality of service, providers reputation, and budget preferences. We also discuss the security functionality and reveal our observations on the utility and users benefits from using our system. Our approach allows users to avoid vendor lock-in, and reduce significantly the cost of switching providers.

1 Introduction

Cloud Computing is a concept of utilizing computing as an on-demand service. It fosters operating and economic efficiencies and promises to cause a significant change in business. Using computing resources as pay-as-you-go model enables service users to convert fixed IT cost into a variable cost based on actual consumption. Therefore, numerous authors argue for the benefits of cloud computing focusing on the economic value [11], [6].

Among available cloud offerings, storage services reveal an increasing level of market competition. According to iSuppli [9] global cloud storage revenue is set to rise to $5 billion in 2013, up from $1.6 billion in 2009. One reason is the ever increasing amount of data which is supposed to outpace the growth of storage capacity. Currently, it is very difficult to estimate the actual future volume of data but there are different estimates being published. According to IDC review [14], the amount of digital information created and replicated is estimated to surpass 3 zettabytes by the

J.J. Park et al. (Eds.): GPC 2013, LNCS 7861, pp. 91–102, 2013.

end of 2012. This amount is supposed to more than double in the next two years. In addition, the authors estimate that today there is 9 times more information available than was available five years ago.

However, for a customer (service) to depend on solely one cloud storage provider (in the following provider) has its limitations and risks. In general, vendors do not provide far reaching security guarantees regarding the data retention. Users have to rely on effectiveness and experience of vendors in dealing with security and intrusion detection systems. For missing guarantees service users are merely advised to encrypt sensitive content before storing it on the cloud. Placement of data in the cloud removes many of direct physical controls that a data owner has over data. So there is a risk that service provider might share corporate data with a marketing company or use the data in a way the client never intended. Further, customers of a particular provider might experience vendor lock-in. In the context of cloud computing, it is a risk for a customer to become dependent on a provider for its services. Common pricing schemes foresee charging for inbound and outbound transfer and requests in addition to hosting the actual data. Changes in features or pricing scheme might motivate a switch from one storage service to another. However, because of the data inertia, customers may not be free to select the optimal vendor due to immense costs associated with a switch of one provider to another. The obvious solution is to make the switching and data placement decisions at a finer granularity then all-or-nothing. This could be achieved by replicating corporate data to multiple storage providers. Such an approach implies significant higher storage and bandwidth costs without taking into account the security concerns regarding the retention of data.

A more economical approach which is presented in this paper is to separate data into unrecognizable slices, which are distributed to providers - whereby only a subset of the nodes needs to be available in order to reconstruct the original data. This is indeed very similar to what has been done for years at the level of file systems and disks. In our work we use RAID-like (Redundant Array of Independent Disks) techniques to overcome the mentioned limitations of cloud storage in the following way:

1. **Security.** The provider might be trustworthy, but malicious insiders represent a well known security problem. This is a serious threat for critical data such as medical records, as cloud provider staff has physical access to the hosted data. One solution might be to encrypt data before the transmission to providers and to decrypt data when receiving those. This requires users to handle the distribution of cryptographic keys when the data needs to be accessed by different users. For each potential customer, it is both expensive and time consuming to handle these security and usability concerns. We tackle the aforementioned problem by encrypting and encoding the original data and later by distributing the fragments transparently across multiple providers. This way, none of the storage vendors is in an absolute possession of the client's data. Moreover, the usage of enhanced erasure algorithms enables us to improve the storage efficiency and thus also to reduce the total costs of the solution.
2. **Service Availability.** Management of computing resources as a service by a single company implies the risk of a single point of failure. This failure depends on many

factors such as financial difficulties (bankruptcy), software or network failure, etc. However, even if the vendor runs data centers in various geographic regions using different network providers, it may have the same software infrastructure. Therefore, a failure in the software in one center will affect all the other centers, hence affecting the service availability. In July 2008, for instance, Amazon storage service S3 was down for 8 hours because of a single bit error [25]. Our solution addresses this issue by storing the data on several clouds - whereby no single entire copy of the data resides in one location, and only a subset of providers needs to be available in order to reconstruct the data.

3. **Reliability.** Any technology can fail. According to a study conducted by Kroll Ontrack[1] 65 percent of businesses and other organizations have frequently lost data from a virtual environment. A number that is up by 140 percent from just last year. Admittedly, in the recent times, no spectacular outages were observed. Nevertheless failures do occur. For example, in October 2009 a subsidiary of Microsoft, Danger Inc., lost the contracts, notes, photos, etc. of a large number of users of the Sidekick service [20]. Most of the data could be recovered within a few weeks, but the users of Ma.gnolia[2] were not so lucky in February of the same year, when the company lost half a terabyte of data [17]. We deal with the problem by using erasure algorithms to separate data into packages, thus enabling the application to retrieve data correctly even if some of the providers corrupt or lose the entrusted data.
4. **Data lock-in.** By today there are no standards for APIs for data import and export in cloud computing. This limits the portability of data and applications between providers. For the customer this means that he cannot seamlessly move the service to another provider if he becomes dissatisfied with the current provider. This could be the case if a vendor increases the fees, goes out of business, or degrades the quality of the provided services. As stated above, our solution does not depend on a single service provider. The data is balanced among several providers taking into account user expectations regarding the price and availability of the hosted content. Moreover, with erasure codes we store only a fraction of the total amount of data on each cloud provider. In this way, switching one provider for another costs merely a fraction of what it would be otherwise.

The main contribution of this paper is: we present a design of an application that can be used to overcome the limitations of individual clouds by using encryption, erasure codes and by integrating various cloud storage providers.

2 Architecture Overview

The ground of our approach is to find a balance between benefiting from the cloud's nature of pay-per-use and ensuring the security of the company's data. The goal is to achieve such a balance by distributing corporate data among multiple storage

[1] http://www.krollontrack.com/resource-library/case-studies/
[2] http://gnolia.com/

providers, automizing big part of the selection process of a cloud provider, and removing the auditing and administrating responsibility from the customer's side. As mentioned above, the basic idea is not to depend on solely one storage provider but to spread the data across multiple providers using redundancy to tolerate possible failures. The approach is similar to a service-oriented version of RAID. While RAID manages sector redundancy dynamically across hard-drives, our approach manages file distribution across cloud storage providers. RAID 5, for example, stripes data across an array of disks and maintains parity data that can be used to restore the data in the event of disk failure. We carry the principle of the RAID-technology to cloud infrastructure. In order to achieve our goal we foster the usage of erasure coding technics (see 3.3). This enables us to tolerate the loss of one or more storage providers without suffering any loss of content [26], [13]. Our architecture includes the following main components:

- **User Interface Module.** The interface presents the user a cohesive view on the data and available features. Here users can manage their data and specify requirements regarding the data retention (quality of service parameters).
- **Resource Management Module.** This system component is responsible for an intelligent deployment of data based on the user's requirements.
- **Data Management Module.** This component handles data management on behalf of the resource management module.

Interested readers will find more background information in our previous work [24],[21]. The system has a number of core components that contain the logic and management layers required to encapsulate the functionality of different storage providers. The next section gives an overview on the implementation of our system on a more detailed level.

3 Design

Any application needs a model of storage, a model of computation and a model of communication. In this section we describe how we achieve the goal of the consistent, unified view on the data management system to the end-user. The web portal is developed using Grails, JNI and C technologies, with a MySQL back-end to store user accounts, current deployments, meta data, and the capabilities and pricing of cloud storage providers. Keeping the meta data locally ensures that no individual provider will have access to stored data. In this way, only users that have authorization to access the data will be granted access to the shares of (at least) k different clouds and will be able to reconstruct the data. Further, our implementation makes use of AES for symmetric encryption, SHA-1 and MD5 for cryptographic hashes and an improved version of Jerasure library [18] for using the Cauchy-Reed-Solomon and Liberation erasure codes. Our system communicates with providers via "storage connectors", which are discussed further in this section.

3.1 Service Interface

The graphical user interface provides two major functionalities to an end-user: data administration and specification of requirements regarding the data storage. Interested readers are directed to our previous work [22] which gives a more detailed background on the identification of suitable cloud providers in our approach. In short, the user interface enables users to specify their requirements (regarding the placement and storage of user's data) manually in form of options, for example:

- **budget-oriented** content deployment (based on the price model of available providers)
- data placement based on **quality of service parameters** (for example availability, throughput or average response time)
- storage of data based on **geographical regions** of the user's choice. The restriction of data storage to specific geographic areas can be reasonable in the case of legal restrictions.

3.2 Storage Repositories

Cloud Storage Providers. Cloud storage providers are modeled as a storage entity that supports six basic operations, shown in table 1. We need storage services to support not more than the aforementioned operations. Further, the individual providers are not trusted. This means that the entrusted data can be corrupted, deleted or leaked to unauthorized parties [16]. This fault model encompasses both malicious attacks on a provider and arbitrary data corruption like the Sidekick case (section 1). The protocols require n = k + m storage clouds, at most m of which can be faulty. Present-day, our prototypical implementation supports the following storage repositories: Amazons S3 (in all available regions: US west and east coast, Ireland, Singapore and Tokyo), Box, Rackspace Cloud Files, Azure, Google Cloud Storage and Nirvanix SND. Further providers can be easily added.

Service Repository. At the present time, the capabilities of storage providers are created semi-automatically based on an analysis of corresponding SLAs which are usually written in a plain natural language [5]. Until now the claims stated in SLAs need to be translated into WSLA statements and updated manually (interested readers will find more background information in our previous work [22]). Subsequently the formalized information is imported into a database of the system component named service repository. The database tracks logistical details regarding the capabilities of storage services such as their actual pricing, SLA offered, and physical locations. With this, the service repository represents a pool with available storage services.

Matching. The selection of storage services for the data distribution occurs based on user preferences set in the user interface. After matching user requirements and provider capabilities, we use the reputation of the providers to produce the final list of potential providers to host parts of the user's data. A provider's reputation holds the

Table 1. Storage connector functions

Function	Description
create(ContainerName)	creates a container for a new user
write(ContainerName, ObjectName)	writes a data object to a user container
read(ContainerName, ObjectName)	reads the specified data object
list(ContainerName)	list all data objects of the container
delete(ContainerName, ObjectName)	removes the data object from the container
getDigest(ContainerName, ObjectName)	returns the hash value of the specified data object

details of his historical performance plus his ratings in the service registries and is saved in a Reputation Object (introduced in our previous work [3], [2], [4]). By reading this object, we know a provider's reputation concerning each performance parameter (e.g. has high response time, low price). With this information the system creates a prioritized list of repositories for each user. In general, the number of storage repositories needed to ensure data striping depends on a user's cost expectations, availability and performance requirements. The total number of repositories is limited by the number of implemented storage connectors.

3.3 Data Management

Data Model. In compliance with [1] we mimic the data model of Amazon's S3 by the implementation of our encoding and distribution service. All data objects are stored in containers. A container can contain further containers. Each container represents a flat namespace containing keys associated with objects. An object can be of an arbitrary size, up to 5 gigabytes (limited by the supported file size of cloud providers). Objects must be uploaded entirely, as partial writes are not allowed as opposed to partial reads. Our system establishes a set of n repositories for each data object of the user. These represent different cloud storage repositories (see figure 1).

Encoding. Upon receiving a write request the system splits the incoming object into k data fragments of an equal size - called chunks. These k data packages hold the original data. In the next step the system adds m additional packages whose contents are calculated from the k chunks, whereby k and m are variable parameters [18]. This means, that the act of encoding takes the contents of k data packages and encodes them on m coding packages. In turn, the act of decoding takes some subset of the collection of n = k + m total packages and from them recalculates the original data. Any subset of k chunks is sufficient to reconstruct the original object of size s [19]. The total size of all data packets (after encoding) can be expressed with the following equation: $\left(\frac{s}{k} * k\right) + \left(\frac{s}{k} * m\right) = s + \left(\frac{s}{k} * m\right) = s * \left(1 + \frac{m}{k}\right)$. With this, the usage of erasure codes increases the total storage by a factor of m k . Summarized, the overall overhead depends on the file size and the defined m and k parameters for the erasure configuration. Figure 2 visualizes the performance of our application using different

erasure configurations. Competitive storage providers claim to have SLAs ranging from 99% to 100% uptime percentages for their services. Therefore choosing m = 1 to tolerate one provider outage or failure at time will be sufficient in the majority of cases. Thus, it makes sense to increase k and spread the packages across more providers to lower the overhead costs.

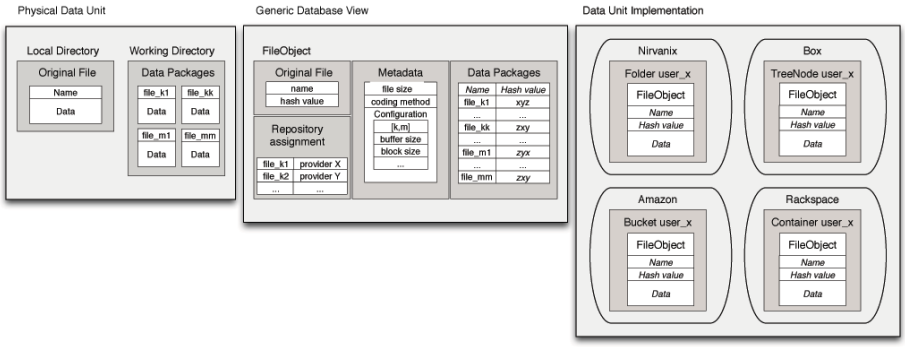

Fig. 1. Data unit model at different abstraction levels. At a physical layer (local directory) each data unit has a name (original file name) and the encoded k+m data packages. In the second level, Cloud-RAID perceives data objects as generic data units in abstract clouds. Data objects are represented as data units with the according meta information (original file name, cryptographic hash value, size, used coding configuration parameters m and k, word size etc.). The database table "Repository Assignment" holds the information about particular data packages and their (physical) location in the cloud. In the third level, data objects are represented as containers in the cloud. Cloud-RAID supports various cloud specific constructions (buckets, treenodes, containers etc.).

In the next step, the distribution service makes sure that each encoded data package is sent to a different storage repository. In general, our system follows a model of one thread per provider per data package in such a way that the encryption, decryption, and provider accesses can be executed in parallel.

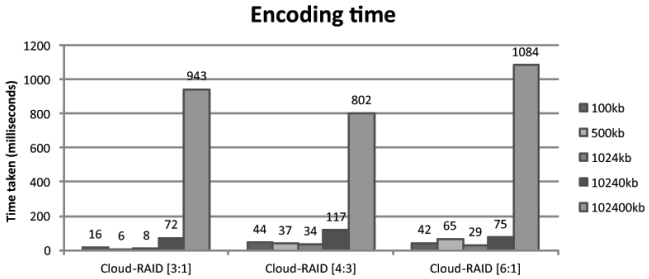

Fig. 2. The average performance of the erasure algorithm with data objects of varying sizes (100kB, 500kB, 1MB, 10MB and 100MB)

However, most erasure codes have further parameters as for example w, which is word size[3]. In addition, further parameters are required for reassembling the data (original file size, hash value, coding parameters, and the erasure algorithm used). This metadata is stored in a MySQL back-end database after performing a successful write request.

Data Distribution. Each storage service is integrated by the system by means of a storage-service-connector (in the following service-connector). These provide an intermediate layer for the communication between the resource management service (see section 3.4) and storage repositories hosted by storage vendors. This enables us to hide the complexity in dealing with proprietary APIs of each service provider. The basic connector functionality covers operations like creation, deletion or renaming of files and folders that are usually supported by every storage provider. Such a service-connector must be implemented for each storage service, as each provider offers a unique interface to its repository. As discussed earlier in this chapter all accesses to the cloud storage providers can be executed in parallel. Therefore, following the encoding, the system performs an initial encryption of the data packages based on one of the predefined algorithms (this feature is optional).

Reassembling the Data. When the service receives a read request, the service component fetches k from n data packages (according to the list with prioritized service providers which can be different from the prioritized write-list, as providers differ in upload and download throughput as well as in cost structure) and reassembles the data. This is due to the fact, that in the pay-per-use cloud models it is not economical to read all data packages from all clouds. Therefore, the service is supported by a load balancer component, which is responsible for retrieving the data units from the most appropriate repositories. Different policies for load balancing and data retrieving are conceivable as parts of user's data are distributed between multiple providers. A read request can be directed to a random data share or the physically closest service (latency-optimal approach). Another possible approach is to fetch data from service providers that meet certain performance criteria (e.g response time or throughput). Finally, there is a minimal-cost aware policy, which guides user requests to the cheapest sources (cost optimal approach). The latter strategy is implemented as a default configuration in our system. Other more sophisticated features as a mix of several complex criteria (e.g. faults and overall performance history) are under development at present. However, the read optimization has been implemented to save time and costs.

3.4 Resource Management Service

This component tracks each user's actual deployment and is responsible for various housekeeping tasks:

[3] The description of a code views each data package as having w bits worth of data.

1. The service is equipped with a MySQL back-end database to store crucial information needed for deploying and reassembling of users data.
5. Further, it audits and tracks the performance of the participated providers and ensures, that all current deployments meet the corresponding requirements specified by the user.
6. The management component is also responsible for scheduling of not time-critical tasks.

Further details can be found in our previous work [21].

4 Related Work

The main idea underlying our approach is to provide RAID technique at the cloud storage level. In [8] the authors introduce the HAIL (High-Availability Integrity Layer) system, which utilizes RAID-like methods to manage remote file integrity and availability across a collection of servers or independent storage services. The system makes use of challenge-responce protocols for retrievability (POR) [15] and proofs of data possession (PDP) [15] and unifies these two approaches. In comparison to our work, HAIL requires storage providers to run some code whereas our system deals with cloud storage repositories as they are. Further, HAIL does not provide confidentiality guarantees for stored data. In [12] Dabek et al. use RAID-like techniques to ensure the availability and durability of data in distributed systems. In contrast to the mentioned approaches our system focuses on the economic problems of cloud computing described in chapter 1.

Further, in [1] authors introduce RACS, a proxy that spreads the storage load over several providers. This approach is similar to our work as it also employs erasure code techniques to reduce overhead while still benefiting from higher availability and durability of RAID-like systems. Our concept goes beyond a simple distribution of users' content. RACS lacks the capabilities such as intelligent file placement based on users' requirements or automatic replication. In addition to it, the RACS system does not try to solve security issues of cloud storage, but focuses more on vendor lock-in. Therefore, the system is not able to detect any data corruption or confidentiality violations.

The future of distributed computing has been a subject of interest for various researchers in recent years. The authors in [10] propose an architecture for market-oriented allocation of resources within clouds. They discuss some existing cloud platforms from the market-oriented perspective and present a vision for creating a global cloud exchange for trading services. Further, our service acts as an abstraction layer between service vendors and service users automatising data placement processes. In fact, our approach enables cloud storage users to place their data on the cloud based on their security policies as well as quality of service expectations and budget preferences. Furthermore, the usage of erasure algorithms for data placement is more efficient than a native replication (in terms of storage and costs).

5 Conclusion

In this paper we outlined some general problems of cloud computing such as security, service availability and a general risk for a customer to become dependent on a service provider. In the course of the paper we demonstrated how our system deals with the mentioned concerns. In a nutshell, we stripe users' data across multiple providers while integrating with each storage provider via appropriate service-connectors. These connectors provide an abstraction layer to hide the complexity and differences in the usage of storage services.

We use erasure code techniques for striping data across multiple providers. The first experiments proved, that given the speed of current disks and CPUs, the libraries used are fast enough to provide good performance, reliable storage system. The average performance overhead caused by data encoding is less than 2% of the amount of time for data transfer to a cloud provider [23]. With this, encoding is dominated by the transmission times and can be neglected. Here, the storage overhead can be varied to achieve higher availability values depending on user requirements. It is up to each individual user to decide whether the additional cost caused by data encoding with higher availability due to determination of higher m parameter are justified. By spreading users data across multiple clouds our approach enables users to avoid the risk of data lock-in and provide a low-level protection even without using security functionality.

However, additional storage offerings are expected to become available in the next few years. Due to the flexible and adaptable nature of our approach, we are able to support any changes in existing storage services as well as incorporating support for new providers as they appear.

6 Future Work

In the last month, we deployed your application using seven commercial cloud storage repositories in different countries in order to conduct a comprehensive test of our system. This includes the predictability and sufficiency of response time and throughput, the overall performance as well as the validation of file consistency.

The results of the experiment are being analysed currently an will be addressed in our next publication. Whilst our system is still under development at present, we will have to use the results of the conducted experiment to improve the overall performance and reliability. This includes for instance the predictability and sufficiency of response time and throughput as well as the validation of file consistency.

In the next step in the development of our registry service we will have to look at ways in which we are able to verify that providers have retained data without retrieving it from the storage repositories and without having to access the entire data. Reading an entire archive, even periodically, is expensive in upload and download costs and limits the scalability of networks. Existing approaches as PDP [7] require service providers to run some code, which is not suitable with our solution.

In addition, we are also planning to implement more service connectors and thus to integrate additional storage services. Any extra storage resource improves the performance and responsiveness of our system for end-users.

References

1. Abu-Libdeh, H., Princehouse, L., Weatherspoon, H.: Racs: A case for cloud storage diversity. In: SoCC 2010 (June 2010)
2. Alnemr, R., Bross, J., Meinel, C.: Constructing a context-aware service-oriented reputation model using attention allocation points. In: Proceedings of the IEEE International Conference on Service Computing, SCC 2009 (2009)
3. Alnemr, R., Meinel, C.: Getting more from reputation systems: A context-aware reputation framework based on trust centers and agent lists. In: International Multi-Conference on Computing in the Global Information Technology (2008)
4. Alnemr, R., Schnjakin, M., Meinel, C.: Towards context-aware service-oriented semantic reputation framework. In: International Joint Conference of IEEE TrustCom/IEEE ICESS/FCST, pp. 362–372 (2011)
5. Amazon. Amazon ec2 service level agreement (2009) (online)
6. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009, EECS Department, University of California, Berkeley (2009)
7. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. Cryptology ePrint Archive, Report 2007/202 (2007)
8. Bowers, K.D., Juels, A., Oprea, A.: Hail: A high-availability and integrity layer for cloud storage. In: CCS 2009 (November 2009)
9. Burt, J.: Future for cloud computing looks good, report says (2009) (online)
10. Buyya, R., Yeo, C.S., Venugopal, S.: Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In: Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications (August 2008)
11. Carr, N.: The Big Switch. Norton (2008)
12. Dabek, F., Kaashoek, M.F., Karger, D., Morris, R., Stoica, I.: Wide-area cooperative storage with cfs. In: ACM SOSP (October 2001)
13. Dingledine, R., Freedman, M.J., Molnar, D.: The free haven project: Distributed anonymous storage service. In: Federrath, H. (ed.) Anonymity 2000. LNCS, vol. 2009, pp. 67–95. Springer, Heidelberg (2001)
14. Gantz, J., Reinsel, D.: Extracting value from chaos (2009) (online)
15. Krawczyk, H.: LFSR-based hashing and authentication. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 129–139. Springer, Heidelberg (1994)
16. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. 4(3), 382–401 (1982)
17. Naone, E.: Are we safeguarding social data? (2009) (online)
18. Plank, J.S., Simmerman, S., Schuman, C.D.: Jerasure: A library in C/C++ facilitating erasure coding for storage applications - Version 1.2. Technical Report CS-08-627, University of Tennessee (August 2008)

19. Rhea, S., Wells, C., Eaton, P., Geels, D., Zhao, B., Weatherspoon, H., Kubiatowicz, J.: Maintenance free global storage in oceanstore. IEEE Internet Computing (September 2001)
20. Sarno, D.: Microsoft says lost sidekick data will be restored to users. Los Angeles Times (October 2009)
21. Schnjakin, M., Alnemr, R., Meinel, C.: A security and high-availability layer for cloud storage. In: Chiu, D.K.W., Bellatreche, L., Sasaki, H., Leung, H.-f., Cheung, S.-C., Hu, H., Shao, J. (eds.) WISE Workshops 2010. LNCS, vol. 6724, pp. 449–462. Springer, Heidelberg (2011)
22. Schnjakin, M., Alnemr, R., Meinel, C.: Contract-based cloud architecture. In: Proceedings of the Second International Workshop on Cloud Data Management, CloudDB 2010, pp. 33–40. ACM, New York (2010)
23. Schnjakin, M., Korsch, D., Schoenberg, M., Meinel, C.: Implementation of a secure and reliable storage above the untrusted clouds. In: Proceedings of 8th International Conference on Computer Science and Education, ICCSE 2013 (to appear in April 2013)
24. Schnjakin, M., Meinel, C.: Platform for a secure storage-infrastructure in the cloud. In: Proceedings of the 12th Deutscher IT-Sicherheitskongress, Sicherheit 2011 (2011)
25. The Amazon S3 Team. Amazon s3 availability event: July 20, 2008 (2008) (online)
26. Weatherspoon, H., Kubiatowicz, J.D.: Erasure coding vs. Replication: A quantitative comparison. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 328–337. Springer, Heidelberg (2002)