COMPLYING WITH DATA HANDLING REQUIREMENTS IN CLOUD STORAGE SYSTEMS

Priti Kandewar[1], S Sravanthi[2], K Krishna[3], D Srinivas[4], Dr. Santhoshkumar[5], S. Bavan kumar[6]
[1,2,3,4,6]Asst. Professor, Dept of CSE, St. Martin's Engineering College, Dulapally, Secunderabad, TS, India,
[5]Associ. Professor, Dept of CSE, St. Martin's Engineering College, Dulapally, Secunderabad, TS, India

ABSTRACT—In past years, cloud storage systems saw an enormous rise in usage. However, despite their popularity and importance as underlying infrastructure for more complex cloud services, today‘s cloud storage systems do not account for compliance with regulatory, organizational, or contractual data handling requirements by design. Since legislation increasingly responds to rising data protection and privacy concerns, complying with data handling requirements becomes a crucial property for cloud storage systems. We present PRADA, a practical approach to account for compliance with data handling requirements in key-value based cloud storage systems. To achieve this goal, PRADA introduces a transparent data handling layer, which empowers clients to request specific data handling requirements and enables operators of cloud storage systems to comply with them. We implement PRADA on top of the distributed database Cassandra and show in our evaluation that complying with data handling requirements in cloud storage systems is practical in real-world cloud deployments as used for microblogging, data sharing in the Internet of Things, and distributed email storage.

I. INTRODUCTION

NOW A DAYS, many web services outsource the storage of data to cloud storage systems. While this offers multiple benefits, clients and lawmakers frequently insist that storage providers comply with different data handling requirements (DHRs), ranging from restricted storage locations or durations [1], [2] to properties of the storage medium such as full disk encryption [3], [4]. However, cloud storage systems do not support compliance with DHRs today. Instead, the selection of storage nodes is primarily optimized towards reliability, availability, and performance, and thus mostly ignores the demand for DHRs. Even worse, DHRs are becoming increasingly diverse, detailed, and difficult to check and enforce [5], while cloud storage systems are becoming more versatile, spanning different continents [6] or infrastructures [7], and even second-level providers [8]. Hence, clients cannot ensure compliance with DHRs when their data is outsourced to cloud storage systems. This apparent lack of control is not merely an academic problem. Since customers have no influence on the treatment of their data in today‘s cloud storage systems, a large set of customers cannot benefit from the advantages offered by the cloud. The Intel IT Center surveys [9] among 800 IT professionals, that 78% of organizations have to comply with regulatory mandates. Again, 78% of organizations are concerned that cloud offers are unable to meet their requirements. In consequence, 57% of organizations actually refrain from outsourcing regulated data to the cloud. The lack of control over the treatment of data in cloud storage hence scares away many clients. This especially holds for the healthcare, financial, and government sectors [9]. Supporting DHRs enables these clients to dictate adequate treatment of their data and thus allows cloud storage operators to enter new markets. Additionally, it empowers operators to efficiently handle differences in regulations [10] (e.g., data protection). Although the demand for DHRs is widely acknowledged, practical support is still severely limited [9], [11], [12]. Related work primarily focuses on DHRs while processing data [13], [14], [15], limits itself to location requirements [16], [17], or treats the storage system as a black box and tries to coarsely enforce DHRs from the outside [12], [18], [19]. Practical solutions for supporting arbitrary DHRs when storing data in cloud storage systems are still missing – a situation that is disadvantageous to clients and operators of cloud storage systems.

Contributions.
In this paper, we present PRADA, a general key-value based cloud storage system that offers rich and practical support for DHRs to overcome current compliance limitations. Our core idea is to add one layer of indirection, which flexibly and efficiently routes data to storage nodes according to the imposed DHRs. We demonstrate this approach along classical key-value stores, while our approach also generalizes to more advanced storage systems. Specifically, we make the following contributions:
1) We comprehensively analyze DHRs and the challenges they impose on cloud storage systems. Our analysis shows that a wide range of DHRs exist, which clients and operators of cloud storage systems have to address.
2) We present PRADA, our approach for supporting DHRs in cloud storage systems. PRADA adds an indirection layer on top of the cloud storage system to store data tagged with DHRs only on nodes that fulfill these requirements. Our design of PRADA is

incremental, i.e., it does not impair data without DHRs. PRADA supports all DHRs that can be expressed as properties of storage nodes as well as any combination thereof. As we show, this covers a wide range of actual use cases.
3) We prove the feasibility of PRADA by implementing it for the distributed database Cassandra (we make our implementation available [20]) and by quantifying the costs of supporting DHRs in cloud storage systems. Additionally, we show PRADA‘s applicability in a cloud deployment along three real-world use cases: a Twitter clone storing two million authentic tweets, a distributed email store handling half a million emails, and an IoT platform persisting 1.8 million IoT messages.

SCOPE OF THE PROJECT

We present PRADA, a practical approach to account for compliance with data handling requirements in key-value based cloud storage systems. To achieve this goal, PRADA introduces a transparent data handling layer, which empowers clients to request specific data handling requirements and enables operators of cloud storage systems to comply with them. We implement PRADA on top of the distributed database Cassandra and show in our evaluation that complying with data handling requirements in cloud storage systems is practical in real-world cloud deployments as used for micro blogging, data sharing in the Internet of Things, and distributed email storage.

OBJECTIVE

In this paper, we present PRADA, a general key-value based cloud storage system that offers rich and practical support for DHRs to overcome current compliance limitations. Our core idea is to add one layer of indirection, which flexibly and efficiently routes data to storage nodes according to the imposed DHRs. We demonstrate this approach along classical key-value stores, while our approach also generalizes to more advanced storage systems

II. RELATED WORK

We categorize our discussion of related work by the different types of DHRs they address. In addition, we discuss approaches for providing assurance that DHRs are respected. Distributing storage of data. To enforce storage location requirements, a class of related work proposes to split data between different storage systems. Wuchner et al. [12] and ¨ Cloud Filter [18] add proxies between clients and operators to transparently distribute data to different cloud storage providers according to DHRs, while NubiSave [19] allows combining resources of different storage providers to fulfill individual redundancy or security requirements of clients. These approaches can treat individual storage systems only as black boxes. Consequently, they do not support fine grained DHRs within the database system itself and are limited to a small subset of DHRs. Sticky policies. Similar to our idea of specifying DHRs, the concept of sticky policies proposes to attach usage and obligation policies to data when it is outsourced to third-parties [31]. In contrast to our work, sticky policies mainly concern the purpose of data usage, which is primarily realized using access control. One interesting aspect of sticky policies is their ability to make them ―stick‖ to the corresponding data using cryptographic measures which could also be applied to PRADA. In the context of cloud computing, sticky policies have been proposed to express requirements on the security and geographical location of storage nodes [32]. However, so far it has been unclear how this could be realized efficiently in a large and distributed storage system. With PRADA, we present a mechanism to achieve this goal. Policy enforcement. To enforce privacy policies when accessing data in the cloud, Betge-Brezetz et al. [13] monitor ´ access of virtual machines to shared file systems and only allow access if a virtual machine is policy compliant. In contrast, Itani et al. [14] propose to leverage cryptographic coprocessors to realize trusted and isolated execution environments and enforce the encryption of data. Espling et al. [15] aim at allowing service owners to influence the placement of their virtual machines in the cloud to realize specific geographical deployments or provide redundancy through avoiding co-location of critical components. These approaches are orthogonal to our work, as they primarily focus on enforcing policies when processing data, while PRADA addresses the challenge of supporting DHRs when storing data in cloud storage systems. Location-based storage. Focusing exclusively on location requirements, Peterson et al. [16] introduce the concept of data sovereignty with the goal to provide a guarantee that a provider stores data at claimed physical locations, e.g., based on measurements of network delay. Similarly, LoSt [17] enables verification of storage locations based on a challenge-response protocol. In contrast, PRADA focuses on the more fundamental challenge of realizing the functionality for supporting arbitrary DHRs. Controlling placement of data. Primarily focusing on distributed hash tables, SkipNet [74] enables control over data placement by organizing data mainly based on string names. Similarly, Zhou et al. [75] utilize location-based node identifiers to encode physical topology and hence provide control over data placement at a coarse grain. In contrast to PRADA, these approaches need to modify the identifier of data based on the DHRs, i.e., knowledge about the specific DHRs of data is required to locate it. Targeting distributed object-based storage systems, CRUSH [76] relies on hierarchies and data distribution policies to control placement of data in a cluster. These data distribution policies are bound to a predefined hierarchy and hence cannot offer the same flexibility as PRADA. Similarly, Tenant-Defined Storage [77] enables clients to store their data according to DHRs. However and in contrast to PRADA, all data of one client needs to have the same DHRs. Finally, Swift Analytics [78] proposes to control the placement of data to speed up big data analytics. Here, data can only be put directly on specified nodes without the abstraction provided by PRADA‘s approach of supporting DHRs. Hippocratic databases. Hippocratic databases store data together with a purpose specification [79]. This allows them to enforce the purposeful use of data using access control and to realize data retention after a certain period. Using Hippocratic databases, it is possible to create an auditing framework to check if a database is complying with its data disclosure policies [33]. However, this concept only considers a single database and not a distributed setting where storage nodes have different data handling capabilities. Assurance. To provide assurance that storage operators adhere to DHRs, de Oliveira et al. [80] propose an architecture to automate the monitoring of compliance to DHRs when transferring data. Bacon et al. [34] and Pasquier et al. [5] show that this can also be achieved using information flow control. Similarly, Massonet et al. [41] propose a monitoring and audit logging architecture in which the infrastructure provider and service provider collaborate to ensure data location compliance. These approaches are orthogonal to our work and could be used to verify that operators of cloud storage systems run PRADA in an honest way and error-free.

III. DATA COMPLIANCE IN CLOUD STORAGE

With the increasing demand for sharing data and storing it at external parties [22], obeying with DHRs becomes a crucial challenge for cloud storage systems [11], [12], [23]. To substantiate this claim, we outline our setting and rigorously analyze existing and potentially future DHRs. Based on this, we derive goals that must be reached to adequately support DHRs in cloud storage systems.

Setting

We tackle the challenge of supporting DHR compliance in cloud storage systems which are realized over a set of nodes in different data centers [24]. To explain our approach in a simple yet general setting, we assume that data is addressed by a distinct key, i.e., a unique identifier for each data item. Key-value based cloud storage systems [25], [26], [27] provide a general, good starting point, since they are widely used and their underlying principles have been adopted in more advanced cloud storage systems [28], [29], [30]. We discuss how our approach can be applied to other types of cloud storage systems in Section 11. As a basis for our discussion, we illustrate our setting in Figure 1. Clients (end users and companies) insert data into Compliance with DHRs has to be realized by the operator of the cloud storage system. Only the operator knows about the characteristics of the storage nodes and can thus make the ultimate decision on which node to store a specific data item. Different works exist that propose cryptographic guarantees [14], accountability mechanisms [33], information flow control [5], [34], or virtual proofs of physical reality [35] to relax trust assumptions on the operator, i.e., providing the client with assurance that DHRs are (strictly) adhered to. Our goals are different: Our main aim is for functional improvements of the status quo. Thus, these works are orthogonal to our approach and can possibly be combined if the operator is not sufficiently trusted.

Data Handling Requirements

We analyze DHRs from client and operator perspective and identify common classes, as well as the need to support also future and unforeseen requirements. Client perspective. DHRs involve constraints on the storage, processing, distribution, and deletion of data in cloud storage. These constraints follow from legal (laws and regulations) [36], [37], contractual (standards and specifications) [38], or intrinsic requirements (user‘s or company‘s individual privacy requirements) [39], [40]. Especially for businesses, compliance with legal and contractual obligations is important to avoid serious (financial) consequences [41]. Location requirements relate to the storage location of data. On one hand, these requirements address concerns raised when data is stored outside of specified legislative boundaries [2], [11]. The EU‘s General Data Protection Regulation [37], e.g., forbids the storage of personal data in jurisdictions with an insufficient level of privacy protection. Also other legislation, besides data protection law, can impose restrictions on the storage location. German tax legislation, e.g., forbids the storage of tax data outside of the EU [23]. On the other hand, clients, especially corporations, can impose location requirements. To increase robustness against outages, a company might demand to store replicas of their data on different continents [39]. Furthermore, an enterprise could require that sensitive data is not stored at a competitor for fear of accidental leaks or deliberate breaches [40].

Goals

Our analysis of real-world demands for DHRs based on legislation, business interests, and future trends emphasizes the importance to support DHRs in distributed cloud storage. We now derive a set of goals that any approach that addresses this challenging situation should fulfill: Comprehensiveness: To address a wide range of DHRs, the approach should work with any DHRs that can be expressed as properties of storage nodes and support the combination of different DHRs. In particular, it should support the requirements in Section 2.2 and be able to adapt to new DHRs. Minimal performance effort: Cloud storage systems are highly optimized and trimmed for performance. Thus, the impact of DHR support on the performance of a cloud storage system should be minimized. Cluster balance: In existing cloud storage systems, the storage load of nodes can easily be balanced to increase performance. Despite having to respect DHRs (and thus limiting the set of possible storage nodes), the storage load of individual storage nodes should be kept balanced. Coexistence: Not all data will be accompanied by DHRs. Hence, data without DHRs should not be impaired by supporting DHRs, i.e., it should be stored in the same way as in a traditional cloud storage system.

IV. SYSTEM OVERVIEW

The problem that has prevented support for DHRs so far stems from the common pattern used to address data in key-value based cloud storage systems: Data is addressed, and hence also partitioned (i.e., distributed to the nodes in the cluster), using a designated key. Yet, the responsible node (according to the key) for storing a data item will often not fulfill the client‘s DHRs. Thus, the challenge addressed in this paper is how to realize compliance with DHRs and still allow for key-based data access. To tackle this challenge, the core idea of PRADA is to add an indirection layer on top of a cloud storage system. We illustrate how we integrate this layer into existing cloud storage systems in Figure 2. If a responsible node cannot comply with stated DHRs, we store the data at a different node, called target node. To enable the lookup of data, the responsible node stores a reference to the target for specific data. As shown in Figure 2, we introduce three new components (capability, relay, and target store) to realize PRADA. Capability store: The global capability store is used to look up nodes that can comply with a specific DHR. Here, the operator of the cloud storage systems specifies for each node in the cluster which DHR properties this node can fulfill. To speed up lookups in the capability store, each node keeps a local copy of the complete capability store. This approach is feasible, as information on DHRs is comparably small and consists of rather static information. Depending on the individual cloud storage system, distributing this information can be realized by pre configuring the capability store for a storage cluster or by utilizing the storage system itself for creating a globally replicated view of node capabilities. We consider all DHRs that describe static properties of a storage node and range from rather simplistic properties such as storage location to more advanced capabilities such as the support for deleting data at a specific date. Relay store: Each node operates a local relay store containing references to data stored at other nodes. More precisely, it contains references to data the node itself is responsible for but does not comply with the DHRs. For each data item, the relay store contains the key of the data, a reference to the node at which the data is stored, and a copy of the DHRs. Target store: Each node stores data that is redirected to it in a target store. The target store operates exactly as a traditional data store, but allows a node to distinguish data that falls under DHRs from data that does not. Alternatives to adding an indirection layer are likely not viable for scalable key-value based cloud storage systems: Although it is possible to encode very short DHRs in the key used for data access [23], this requires knowledge about DHRs of a data item to compute the key for accessing it and disturbs load balancing. Alternatively, replication of all relay information on all nodes of a cluster allows nodes to derive relay information locally. This, however, severely impacts scalability of the cloud storage system and reduces the total storage amount to the limited storage space of single nodes. Integrating PRADA into a cloud storage system requires us to adapt storage operations (e.g., creating and updating data) and to reconsider replication, load balancing, and failure recovery strategies in the presence of DHRs. In the following, we describe how we address these issues

V. CLOUD STORAGE OPERATIONS

The most important modifications of PRADA involve the CRUD (create, read, update, delete) operations. In the following, we describe how we integrate PRADA into the CRUD operations of our cloud storage model (cf. Section 2.1). We assume that queries are processed on behalf of the client by one of the nodes in the cluster, the coordinator node (as common in cloud storage systems [26]). Each node of the cluster can act as coordinator for a query and clients use the capability store to select a coordinator that complies with the requested DHRs. If no DHRs need to be considered, clients select a coordinator based on performance metrics such as proximity. For reasons of clarity, we postpone the discussion of the impact of different replication factors and load balancing decisions to Section 5 and 6, respectively.

VI. REPLICATIONS

Cloud storage systems employ replication to realize high availability and data durability [26]: Instead of storing a data item only on one node, it is stored on r nodes (typically, with a replication factor $1 \leq r \leq 3$). In key-value based storage systems, the r nodes are chosen based on the key of data (see Section 3). When complying with DHRs, we cannot use the same replication strategy. In the following, we thus detail how PRADA realizes replication instead. Creating data. Instead of selecting only one target, the coordinator picks r targets out of the eligible nodes. The coordinator sends the data to all r targets and the list of all r targets to the r responsible nodes (according to the replication strategy of the cloud storage system). Consequently, each of the r responsible nodes knows about all r targets and can update its relay store accordingly.

Reading data.

To process a read request, the coordinator forwards the read request to all responsible nodes. A responsible node that receives a read request for data it does not store locally looks up the targets in its relay store and forwards the read request to one of the r target nodes. To ensure that each target node receives a request, each responsible node uses the same consistent mapping between responsible and target nodes which is computed based on node identifiers. Each target that receives a read request sends the requested data to the coordinator for this request. If a read query is reissued due to a failure (cf. Section 7), each responsible node will forward the request to all r target nodes to increase reliability. Impact on reliability. To successfully process a query in PRADA, it suffices if one responsible node and one target node are reachable. Thus, PRADA can tolerate the failure of up to r − 1 responsible nodes and up to r − 1 target nodes.

VII. IMPLEMENTATIONS

For the practical evaluation of our approach, we fully implemented PRADA on top of Cassandra [26] (our implementation is available under the Apache License [20]). Cassandra is a distributed database that is actively employed as a key-value based cloud storage system by more than 1500 companies with deployments of up to 75 000 nodes [53] and offers high scalability even over multiple data centers [54], which makes it especially suitable for our scenario. Cassandra also implements advanced features that go beyond simple key-value storage such as column-orientation and queries over ranges of keys, which allows us to showcase the flexibility and adaptability of our design. Data in Cassandra is divided into multiple logical databases, called key spaces. A key space consists of tables which are called column families and contain rows and columns. Each node knows about all other nodes and their ranges of the hash table. Cassandra uses the gossiping protocol Scuttlebutt [50] to efficiently distribute this knowledge as well as to detect node failure and exchange node state, e.g., load information. Our implementation is based on Cassandra 2.0.5, but our design conceptually also works with newer versions. Information stores. PRADA relies on three information stores: the global capability store as well as relay and target stores (cf. Section 3). We implement these as individual key spaces in Cassandra as detailed in the following. First, we realize the global capability store as a key space that is globally replicated among all nodes (i.e., each node stores a full copy of the capability store to improve performance of create operations) initialized at the same time as the cluster with deployments of up to 75 000 nodes [53] and offers high scalability even over multiple data centers [54], which makes it especially suitable for our scenario. Cassandra also implements advanced features that go beyond simple key-value storage such as column-orientation and queries over ranges of keys, which allows us to showcase the flexibility and adaptability of our design. Data in Cassandra is divided into multiple logical databases, called key spaces. A key space consists of tables which are called column families and contain rows and columns. Each node knows about all other nodes and their ranges of the hash table. Cassandra uses the gossiping protocol Scuttlebutt [50] to efficiently distribute this knowledge as well as to detect node failure and exchange node state, e.g., load information. Our implementation is based on Cassandra 2.0.5, but our design conceptually also works with newer versions.

Information stores.

PRADA relies on three information stores: the global capability store as well as relay and target stores (cf. Section 3). We implement these as individual key spaces in Cassandra as detailed in the following. First, we realize the global capability store as a key space that is globally replicated among all nodes (i.e., each node stores a full copy of the capability store to improve performance of create operations) initialized at the same time as the cluster. On this key space, we create a column family for each DHR type (as introduced in Section 2.2). When a node joins the cluster, it inserts all DHR properties it supports for each DHR type (as locally configured by operator of the cloud storage system) into the corresponding column family. This information is then automatically replicated to all other nodes in the cluster by the replication strategy of the corresponding key space. For each regular key space of the database, we additionally create a corresponding relay store and target store as key spaces. Here, the relay store inherits the replication factor and replication strategy from the corresponding regular key space to achieve replication for PRADA as detailed in Section 5, i.e., the relay store will be replicated in exactly the same way as the regular key store. Hence, for each column family in the corresponding key space, we create a column family in the relay key space that acts as the relay store. We follow a similar approach for realizing the target store, i.e., we create for each key space a corresponding key space to store actual data. For each column family in the original key space, we create an exact copy in the target key space to act as the target store. However, to ensure that DHRs are adhered to, we implement a DHR-agnostic replication mechanism for the target store and use the relay store to address data. While the global capability store is created when the cluster is initiated, relay and target stores have to be created whenever a new key space and column family is created, respectively. To this end, we hook into Cassandra‘s Create Key space Statement class for detecting requests for creating key spaces and column families and subsequently initialize the corresponding relay and target stores.

Creating data and load balancing

To allow clients to specify their DHRs when inserting or updating data, we support the specification of arbitrary DHRs in textual form for INSERT requests (cf. Section 2.1). To this end, we add an optional postfix WITH REQUIREMENTS to INSERT statements by extending the grammar from which parser and lexer for CQL3 [55], the SQL-like query language of Cassandra, are generated using ANTLR [56]. Using the WITH REQUIREMENTS statement, arbitrary DHRs can be specified separated by the keyword AND, e.g., INSERT ... WITH REQUIREMENTS location = { ‘DE‘, ‘FR‘, ‘UK‘ } AND encryption = { ‘AES-256‘ }. In this example, any node located in Germany, France, or the United Kingdom that supports AES-256 encryption is eligible to store the inserted data. This approach enables users to specify any DHRs covered by our formalized model of DHRs (cf. Section 2.2). To detect and process DHRs in create requests (cf. Section 4), we extend Cassandra‘s Query Processor, specifically its get Statement method for processing INSERT requests. When processing requests with DHRs (specified using the WITH REQUIREMENTS statement), we base our selection of eligible nodes on the global capability store. Nodes are eligible to store data with a given set of DHRs if they provide at least one of the specified properties for each requested type (e.g., one out of multiple permitted locations). We prioritize nodes that Cassandra would pick without DHRs, as this speeds up reads for the corresponding key later on, and otherwise choose nodes according to our load balancer (cf. Section 6). Our load balancing implementation relies on Cassandra‘s gossiping mechanism [26], which maintains a map of all nodes together with their corresponding loads. We access this information using Cassandra‘s get Load Info method and extend the load information with local estimators for load changes. Whenever a node sends a create request or stores data itself, we update the corresponding local estimator with the size of the inserted data. To this end, we hook into the methods that are called when data is modified locally or forwarded to other nodes, i.e., the corresponding methods in Cassandra‘s Modification Statement, Row Mutation Verb Handler, and Storage Proxy classes as well as our methods for processing requests with DHRs.

Reading data

To allow reading redirected data as described in Section 4, we modify Cassandra‘s Read Verb Handler class for processing read requests at the responsible node. This handler is called whenever a node receives a read request from the coordinator and allows us to check whether the current node holds a reference to another target node for the requested key by locally checking the corresponding column family within the relay store. If no reference exists, the node continues with a standard read operation. Otherwise, the node forwards a modified read request to one deterministically selected target node (cf. Section 5) using Cassandra‘s sendOneWay method, in which it requests the data from the respective target on behalf of the coordinator. Subsequently, the target nodes send the data directly to the coordinator node (whose identifier is included in the request). To correctly resolve references to data for which the coordinator of a query is also the responsible node, we additionally add corresponding checks to the Local Read Runnable subclass of Storage Proxy.

VIII. RESULTS

DATA HANDLING REQUIREMENTS

All Users Details

FIG:1 ALL USERS DETAILS

DATA HANDLING REQUIREMENTS

Accept New Files Details

FIG 2: ACCEPT NEW FILES DETAILS

DATA HANDLING REQUIREMENTS

Deleted Files

[illegible]	[illegible]	Email	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

FIG 3: DELETED FILES

IX. DISCUSSION AND CONCLUSION

Accounting for compliance with data handling requirements (DHRs), i.e., offering control over where and how data is stored in the cloud, becomes increasingly important due to legislative, organizational, or customer demands. Despite these incentives, practical solutions to address this need in existing cloud storage systems are scarce. In this paper, we proposed PRADA, which allows clients to specify a comprehensive set of fine-grained DHRs and enables cloud storage operators to enforce them. Our results show that we can indeed achieve support for DHRs in cloud storage systems. Of course, the additional protection and flexibility offered by DHRs comes at a price: We observe a moderate increase for query completion times, while achieving constant storage overhead and upholding a near optimal storage load balance even in challenging scenarios. Importantly, however, data without DHRs is not impaired by PRADA. When a responsible node receives a request for data without DHRs, it can locally check that no DHRs apply to this data: For create requests, the INSERT statement either contains DHRs or not, which can be checked efficiently and locally. In contrast, for read, update, and delete requests, PRADA performs a simple and local check whether a reference to a target node for this data exists. The overhead for this step is comparable to executing an if statement and hence negligible. Only if a reference exists, which implies that the data was inserted with DHRs, PRADA induces overhead. Our extensive evaluation confirms that, for data without DHRs, PRADA shows the same query completion times, storage overhead, and bandwidth consumption as an unmodified Cassandra system in all considered settings (indistinguishable results for Cassandra and PRADA* in Figures 5 to 8.) Consequently, clients can choose (even at a granularity of individual data items), if DHRs are worth a modest performance decrease. PRADA‘s design is built upon a transparent indirection layer, which effectively handles compliance with DHRs. This design decision limits our solution in three ways. First, the overall achievable load balance depends on how well the nodes‘ capabilities to fulfill certain DHRs matches the actual DHRs requested by the clients. However, for a given scenario, PRADA is able to achieve nearly optimal load balance as shown in Figure 10. Second, indirection introduces an overhead of 0.5 round-trip times for reads, updates, and deletes. Further reducing this overhead is only possible by encoding some DHRs in the key used for accessing data [23], but this requires everyone accessing the data to be in possession of the DHRs, which is unlikely. A fundamental improvement could be achieved by replicating all relay information to all nodes of the cluster, but this is viable only for small cloud storage systems and does not offer scalability. We argue that indirection can likely not be avoided, but still pose this as an open research question. Third, the question arises how clients can be assured that an operator indeed enforces their DHRs and no errors in the specification of DHRs have occurred. This has been widely studied [16], [33], [41], [80] and the proposed approaches such as audit logging, information flow control, and provable data possession can also be applied to PRADA. While we limit our approach for providing data compliance in cloud storage to key-value based storage systems, the key-value paradigm is also general enough to provide a practical starting point for storage systems that are based on different paradigms. Additionally, the design of PRADA is flexible enough to extend (with some more work) to other storage systems. For example, Google‘s globally distributed database Spanner (rather a multi-version database than a key-value store) allows applications to influence data locality (to increase performance) by carefully choosing keys [28]. PRADA could be applied to Spanner by modifying Spanner‘s approach of directory-bucketed key-value mappings. Likewise, PRADA could realize data compliance for distributed main memory databases, e.g., VoltDB, where tables of data are partitioned horizontally into shards [29]. Here, the decision on how to distribute shards over the nodes in the cluster could be taken with DHRs in mind. Similar adaptations could be performed for commercial products, such as Clustrix [30], that separate data into slices. To conclude, PRADA resolves a situation, i.e., missing support for DHRs, that is disadvantageous to both clients and operators of cloud storage systems. By offering the enforcement of arbitrary DHRs when storing data in cloud storage systems, PRADA enables the use of cloud storage systems for a wide range of clients who previously had to refrain from outsourcing storage, e.g., due to compliance with applicable data protection legislation. At the same time, we empower cloud storage operators with a practical and efficient solution to handle differences in regulations and offer their services to new clients.

REFERENCE

[1] R. Gellman, ―Privacy in the Clouds: Risks to Privacy and Confidentiality from Cloud Computing,‖ World Privacy Forum, 2009.
[2] S. Pearson and A. Benameur, ―Privacy, Security and Trust Issues Arising from Cloud Computing,‖ in IEEE CloudCom, 2010.
[3] United States Congress, ―Gramm-Leach-Bliley Act (GLBA),‖ Pub.L. 106-102, 113 Stat. 1338, 1999.
[4] D. Song et al., ―Cloud Data Protection for the Masses,‖ Computer, vol. 45, no. 1, 2012.
[5] T. F. J. M. Pasquier et al., ―Information Flow Audit for PaaS Clouds,‖ in IEEE IC2E, 2016.
[6] V. Abramova and J. Bernardino, ―NoSQL Databases: MongoDB vs Cassandra,‖ in C3S2E, 2013.
[7] R. Buyya, R. Ranjan, and R. N. Calheiros, ―InterCloud: Utility- Oriented Federation of Cloud Computing Environments for Scaling of Application Services,‖ in ICA3PP, 2010.
[8] D. Bernstein et al., ―Blueprint for the Intercloud - Protocols and Formats for Cloud Computing Interoperability,‖ in ICIW, 2009.
[9] Intel IT Center, ―Peer Research: What‗s Holding Back the Cloud?‖ Tech. Rep., 2012.
[10] D. Catteddu and G. Hogben, ―Cloud Computing – Benefits, Risks and Recommendations for Information Security,‖ European Network and Information Security Agency (ENISA), 2009.
[11] M. Henze, R. Hummen, and K. Wehrle, ―The Cloud Needs Cross- Layer Data Handling Annotations,‖ in IEEE S&P Workshops, 2013.
[12] T. W¨ uchner, S. M¨ uller, and R. Fischer, ―Compliance-Preserving Cloud Storage Federation Based on Data-Driven Usage Control,‖ in IEEE CloudCom, 2013.
[13] S. Betg´e-Brezetz et al., ―End-to-End Privacy Policy Enforcement in Cloud Infrastructure,‖ in IEEE CloudNet, 2013.
[14] W. Itani, A. Kayssi, and A. Chehab, ―Privacy as a Service: Privacy-Aware Data Storage and Processing in Cloud Computing Architectures,‖ in IEEE DASC, 2009.
[15] D. Espling et al., ―Modeling and Placement of Cloud Services with Internal Structure,‖ IEEE Transactions on Cloud Computing, vol. 4, no. 4, 2014.
[16] Z. N. J. Peterson, M. Gondree, and R. Beverly, ―A Position Paper on Data Sovereignty: The Importance of Geolocating Data in the Cloud,‖ in USENIX HotCloud, 2011.
[17] G. J. Watson et al., ―LoSt: Location Based Storage,‖ in ACM CCSW, 2012.
[18] I. Papagiannis and P. Pietzuch, ―CloudFilter: Practical Control of Sensitive Data Propagation to the Cloud,‖ in ACM CCSW, 2012.
[19] J. Spillner, J. M¨ uller, and A. Schill, ―Creating optimal cloud storage systems,‖ Future Generation Computer Systems, vol. 29, no. 4, 2013.
[20] RWTH Aachen University, ―PRADA Source Code Repository,‖ https://github.com/COMSYS/prada.
[21] M. Henze et al., ―Practical Data Compliance for Cloud Storage,‖ in IEEE IC2E, 2017.
[22] P. Samarati and S. De Capitani di Vimercati, ―Data Protection in Outsourcing Scenarios: Issues and Directions,‖ in ACM ASIACSS, 2010.
[23] M. Henze et al., ―Towards Data Handling Requirements-aware Cloud Computing,‖ in IEEE CloudCom, 2013.
[24] A. Greenberg et al., ―The Cost of a Cloud: Research Problems in Data Center Networks,‖ SIGCOMM Comput. Commun. Rev., vol. 39, no. 1, 2008.
[25] G. DeCandia et al., ―Dynamo: Amazon‗s Highly Available Keyvalue Store,‖ in ACM SOSP, 2007.
[26] A. Lakshman and P. Malik, ―Cassandra: A Decentralized Structured Storage System,‖ ACM SIGOPS Operating Systems Review, vol. 44, no. 2, 2010.
[27] M. T. O¨ zsu and P. Valduriez, Principles of Distributed Database Systems, 3rd ed. Springer, 2011.
[28] J. C. Corbett et al., ―Spanner: Google‗s Globally-distributed Database,‖ in USENIX OSDI, 2012.
[29] M. Stonebraker and A. Weisberg, ―The VoltDB Main Memory DBMS,‖ IEEE Data Eng. Bull., vol. 36, no. 2, 2013.
[30] Clustrix, Inc., ―Scale-Out NewSQL Database in the Cloud,‖ http: //www.clustrix.com/.
[31] S. Pearson and M. C. Mont, ―Sticky Policies: An Approach for Managing Privacy across Multiple Parties,‖ Computer, vol. 44, no. 9, 2011.
[32] S. Pearson, Y. Shen, and M. Mowbray, ―A Privacy Manager for Cloud Computing,‖ in CloudCom, 2009.
[33] R. Agrawal et al., ―Auditing Compliance with a Hippocratic Database,‖ in VLDB, 2004.
[34] J. Bacon et al., ―Information Flow Control for Secure Cloud Computing,‖ IEEE Transactions on Network and Service Management, vol. 11, no. 1, 2014.
[35] U. R¨ uhrmair et al., ―Virtual Proofs of Reality and their Physical Implementation,‖ in IEEE S&P, 2015.
[36] United States Congress, ―Health Insurance Portability and Accountability Act of 1996 (HIPAA),‖ Pub.L. 104–191, 110 Stat. 1936, 1996.
[37] ―Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation),‖ L119, 4/5/2016, 2016.
[38] PCI Security Standards Council, ―Payment Card Industry (PCI) Data Security Standard – Requirements

and Security Assessment Procedures, Version 3.1,‖ 2015.
[39] R. Buyya et al., ―Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility,‖ Future Generation Computer Systems, vol. 25, no. 6, 2009.
[40] T. Ristenpart et al., ―Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds,‖ in ACM CCS, 2009.
[41] P. Massonet et al., ―A Monitoring and Audit Logging Architecture for Data Location Compliance in Federated Cloud Infrastructures,‖ in IEEE IPDPS Workshops, 2011.
[42] United States Congress, ―Sarbanes-Oxley Act (SOX),‖ Pub.L. 107–204, 116 Stat. 745, 2002.
[43] A. Mantelero, ―The EU Proposal for a General Data Protection Regulation and the roots of the ‗right to be forgotten',‖ Computer Law & Security Review, vol. 29, no. 3, 2013.
[44] H. A. J¨ager et al., ―Sealed Cloud – A Novel Approach to Safeguard against Insider Attacks,‖ in Trusted Cloud Computing. Springer, 2014.
[45] J. Singh et al., ―Regional clouds: technical considerations,‖ University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CLTR- 863, 2014.
[46] S. Pearson, ―Taking Account of Privacy when Designing Cloud Computing Services,‖ in Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing. IEEE, 2009.
[47] E. Barker, ―Recommendation for Key Management – Part 1: General (Revision 4),‖ NIST Special Publication 800-57, National Institute of Standards and Technology, 2015.
[48] A. Corradi, L. Leonardi, and F. Zambonelli, ―Diffusive Load- Balancing Policies for Dynamic Applications,‖ IEEE Concurrency, vol. 7, no. 1, 1999.
[49] L. Rainie and J. Anderson, ―The Future of Privacy,‖ Pew Research Center, http://www.pewinternet.org/2014/12/18/futureof-privacy/, 2014.
[50] R. van Renesse et al., ―Efficient Reconciliation and Flow Control for Anti-entropy Protocols,‖ in LADIS, 2008.