COMPLYING WITH DATA HANDLING REQUIREMENTS IN CLOUD STORAGE SYSTEMS

Mohd Azeemullah[1], Dr. Jaibir Singh[2]
[1]Research Scholar, Dept. of CSE, OPJS University, Churu.
azeemullah889@gmail.com
[2]Associate Professor and Dean, Dept. of CSE-IoT, OPJS University, Churu.
jaibir729@gmail.com

ABSTRACT

The use of cloud storage systems has significantly increased in recent years. However, despite their widespread use and significance as the foundation for increasingly complex cloud services, current cloud storage systems are not built to take compliance with legal, organizational, or contractual requirements for data management into account. Compliance with data handling regulations becomes an important characteristic for cloud storage systems as government progressively responds to growing data protection and privacy concerns. We describe PRADA, a workable method for ensuring that key-value based cloud storage systems comply with data handling specifications. To do this, PRADA creates a transparent data handling layer that gives clients the ability to request specific data handling needs and gives cloud storage system operators the ability to adhere to them. PRADA is implemented on top of the distributed database Cassandra and we demonstrate in our study that it is feasible in real-world cloud deployments for microblogging, data sharing in the Internet of Things, and distributed email storage to adhere to data handling criteria in cloud storage systems.

Keywords: Cloud Computing, Data Handling, Compliance, Load Balancing, Sharing Data, Cloud Storage Systems, Distributed Databases, Privacy, Public Policy Issues

DOI Number: 10.48047/NQ.2022.20.20.NQ109226 **NeuroQuantology2022;20(20): 2255-2268**

1. INTRODUCTION

Many modern web services now use cloud storage systems to handle their data storage needs. While this has many advantages, customers and legislators frequently demand that storage providers adhere to various data handling requirements (DHRs), which can range from limitations on storage locations or times to features of the storage medium like complete disc encryption. Cloud storage systems do not, however, currently offer DHR compliance. Instead, stability, availability, and performance are prioritised over DHR demand in the selection of storage nodes, which mostly ignores it. Even worse, cloud storage solutions are getting more adaptable, spanning several regions or infrastructures, and even second-level providers, whereas DHRs are growing more diverse, specific, and challenging to audit and enforce. Therefore, clients cannot guarantee adherence to DHRs when Their information is sent to cloud storage platforms.

This apparent lack of self-control extends beyond the classroom. Many consumers are unable to take use of the benefits provided by the cloud because they have little control over how their data is handled in today's cloud storage systems. According to a poll conducted by the Intel IT Center [9] among 800 IT experts, 78% of firms are required to follow legal requirements. Once more, 78% of firms worry that cloud offerings won't be able to satisfy their needs. As a result, 57% of businesses actually don't outsource their regulated data to the cloud. Therefore, many customers avoid using cloud storage due to the lack of control over how their data is handled. For the government, banking, and healthcare sectors in particular, this is true Supporting DHRs gives these customers the power to decide how their data should be handled, opening up new markets for cloud storage providers. Furthermore, it equips operators to effectively manage variations in rules (e.g., data protection). Although the need for DHRs is widely accepted, there is currently very little in the way of real support. Related work either restricts itself to location requirements, approaches the storage system as a black box, or attempts to obtrusively impose DHRs from the outside. Related work largely focuses on DHRs when processing data. Customers and cloud storage system operators suffer from the lack of workable options for supporting arbitrary DHRs while storing data in cloud storage systems.

Our Contributions: In this work, we introduce PRADA, a universal key-value based cloud storage solution that provides DHRs with comprehensive and useful functionality to get beyond current compliance constraints. Our main concept is to introduce one indirection layer that flexible and effectively directs data to storage nodes in accordance with the mandated DHRs. This method is illustrated using traditional key-value stores, but it can also be used to more

sophisticated storage systems. We specifically contribute the following:

1) We thoroughly examine DHRs and the difficulties they present for cloud storage systems. Our investigation reveals that there are numerous DHRs, which users and administrators of cloud storage systems must address.

2) PRADA, our method for assisting DHRs in cloud storage systems, is presented. PRADA includes a a layer of indirection on top of the cloud storage platform to store data marked with DHRs exclusively on nodes meeting these specifications. Our PRADA approach is incremental, therefore data without DHRs are not harmed by it. All DHRs that may be specified as storage node characteristics, alone or in combination, are supported by PRADA. This covers a wide range of real-world use scenarios, as we demonstrate.

3) We evaluate the expenses of supporting DHRs in cloud storage systems and develop PRADA for the distributed database Cassandra (we make our implementation publicly available) to demonstrate the viability of PRADA. In addition, we demonstrate the viability of PRADA in a cloud deployment through three real-world use cases: a replica of Twitter that stores two million real tweets, a distributed email store, and managing 500,000 emails and 1.8 million IoT messages, according to an IoT platform.

Paper structure. In Section 2, we analyze DHRs and derive goals for supporting DHRs in cloud storage systems. We provide an overview of our design in Section 3, before we provide details on individual storage operations (Section 4), replication (Section 5), load balancing (Section 6), and failure recovery (Section 7). Subsequently, we describe our implementation in Section 8 and evaluate its performance and applicability in Section 9and conclude with a discussion in Section 10.

2. DATA COMPLIANCE IN CLOUD STORAGE

Obeying with DHRs becomes a significant difficulty for cloud storage systems [11], [12], and [23] due to the growing demand for data exchange and storage at third parties [22]. We describe our setup and conduct a thorough analysis of current and prospective future DHRs to support this assertion. On the basis of this, we determine the objectives that need to be accomplished in order to support DHRs in cloud storage systems.

2.1 SETTING

In this article, we address the issue of supporting DHR compliance in cloud storage systems that are implemented as a collection of nodes spread across various data centres [24]. We make the assumption that data is addressed by a unique key, i.e., a unique identifier for each data item, in order to describe our technique in a straightforward yet generic context. Given their widespread use and the fact that more sophisticated cloud storage systems [28], [29], and [30] have incorporated their core ideas, key-value based cloud storage systems [25], [26], and [27] offer a broad, good starting point. We talk about how our strategy may be used with various cloud storage platforms.

Figure 1 serves as the foundation for our discussion by illustrating our location. End users and businesses that serve as clients enter data into the cloud storage platform and add DHR annotations on it. These specifications are written in text and are comprehensible to the cloud storage system's operator. Data handling annotations [11], [23] and sticky policies [31, 32] are other names for the process of annotating data with DHRs. Every storage system client has the option to impose unique and different DHRs for every single added data item.

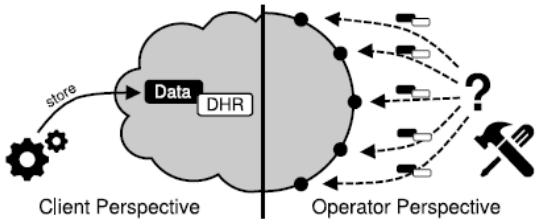

Fig.1: Setting. When clients insert data with DHRs, the operator has to store it only on nodes of the storage system complying with the DHRs.

The cloud storage system's operator must ensure compliance with DHRs. Only the operator can decide which storage node to use to store a certain data item because only they are aware of the storage nodes' attributes. There are other papers that suggest accountability systems [33], information flow control [5], [34], or cryptographic guarantees [14] To reduce reliance on the operator's word, virtual proofs of physical reality [35] are used to reassure clients that DHRs are (strictly) followed. Our objectives diverge: Our primary goal is to make the current situation more functional. As a result, these works are not compatible with our strategy and could be merged if the operator cannot be sufficiently trusted.

2.2 DATA HANDLING REQUIREMENTS:

We examine DHRs from the viewpoints of the client and the operator, identifying common classes as well as the necessity to handle additional, unforeseen requirements.

Client Perspective: DHRs place restrictions on how data is stored, processed, distributed, and deleted from cloud storage. Legal (rules and regulations) [36, [37], contractual (standards and specifications) [38], or intrinsic requirements (users' or companies' own privacy requirements) [39, [40] all result in these limitations. Compliance with legal and contractual duties, particularly for organisations, is critical to avoiding negative (money) outcomes [41]. Location requirements have to do with where the data will be

stored. On the one hand, these standards address issues that are brought up when data is stored outside of authorized legal restrictions [2],[11]. For instance, the EU's General Data Protection Regulation [37] prohibits the storage of personal data in nations that do not provide enough privacy protection. In addition to data protection law, other laws may also put limitations on where data is stored. Germany's tax laws, restricts the storing of tax data outside the EU, for instance [23]. However, clients, particularly businesses, have the power to set location restrictions. A business may request to store copies of its data on many continents to boost robustness against outages [39]. For fear of unintentional leaks or intentional breaches, an organization may also demand that sensitive data not be kept at a rival [40].

Data storage time is constrained by duration requirements. For instance, the Sarbanes-Oxley Act (SOX) [42] mandates that accounting firms hold onto data necessary for audits and reviews for a period of seven years. However, the Payment Card Industry Data Security Standard (PCI DSS) [38] only permits the preservation of cardholder data for as long as is required for business, legal, or regulatory purposes following it must be removed. In the EU and Argentina, a similar strategy known as "the right to be forgotten" is actively being considered and made into law [37], [43]. Further defining how data should be stored are the traits requirements. For instance, before disposing of or utilising a storage medium again, the US Health Insurance Portability and Accountability Act (HIPAA) [36] mandates that health data be safely erased. Similar to this, the Gramm-Leach-Bliley Act (GLBA) [3] mandates appropriate encryption of customer data for the banking and financial services sector. Clients can also decide to exclusively keep their data on volatile [44] or fully encrypted [4] storage in order to safeguard it from theft or confiscation. operator's point of view Support for DHRs offers operators of cloud storage strong commercial advantages because it expands their market and makes regulatory compliance simpler. The distinctive selling proposition that DHRs offer to the untapped market of customers who are currently unable to outsource their data to cloud storage systems due to unfulfillable DHRs provides business incentives [9]. In fact, some well-chosen requirements have already been accommodated by cloud providers. For instance, Google developed the separate "Google Apps for Government" and had it certified at the FISMA-Moderate level, allowing use by US federal agencies [41] in order to be able to offer its services to the US government. Additionally, cloud service providers set up data centres all over the world to meet client location needs [7]. From a different angle, localised clouds, such as the anticipated "Europe-only" clouds [45] seek to improve data governance and control. Additionally, giving customers more control over their data lowers the dangers of losing credibility and goodwill [46].

Independent of particular corporate objectives and incentives, operators must comply with the law. For instance, the business associate agreement of HIPAA [36] mandates that while transmitting electronic health records [1], the operator adhere to the same rules as its clients. Additionally, DHRs must be followed by data controllers from outside the EU who process data coming from the EU, as per the EU's General Data Protection Regulation [37]. Future necessities Just as laws and technology develop and advance over time, DHRs are expected to do the same. Location needs evolved, for example, when cloud storage systems

2.3 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 span several geographical areas. Since it is hard to foresee every potential future change in DHRs, it is essential that DHR support in cloud storage systems be flexible enough to meet changing needs. formalising the requirements for data handling. We build our method on a defined concept of DHRs that also covers yet unanticipated DHRs in order to support future requirements and storage systems. In order to achieve this, we categorise different types of DHRs and take into account various potential attributes that storage nodes (can) support for a given category of DHRs. This enables the computation of the set of eligible nodes for a certain type of DHRs, i.e., the nodes that provide the client's desired attributes.

Storage location is a straightforward illustration of a DHR type. In this illustration, all potential storage locations are included in the attributes, and nodes are only considered eligible if their storage location matches the one that the customers have requested. In a more complex illustration, we take into

account full-disk encryption's security level as the DHR type. The security levels here range from 0 bits (no encryption) to various security bits (such as 192 bits or 256 bits), with more security bits delivering a greater security level [47]. In this scenario, all storage nodes that offer at least the client-requested security level are regarded as being qualified to store the data.

We provide a service by enabling clients to combine various DHR kinds and to specify a set of necessary features (for instance, various storage locations) for each type strong tools for expressing DHRs to people. Section 4 describes how clients can combine various types, and Section 8 describes how we incorporate DHRs into the Cassandra query language.

3. SYSTEM OVERVIEW

Support for DHRs has so far been hindered by a common pattern used to handle data in key-value based cloud storage systems: Using a chosen key, data is addressed and subsequently partitioned (i.e., distributed to the cluster's nodes). However, the client's DHRs are frequently not met by the node that is accountable (as determined by the key) for storing a data item. The problem that needs to be solved in this study is how to achieve DHR compliance while maintaining key-based data access.

The fundamental concept behind PRADA is to build an indirection layer on top of a cloud storage system in order to address this problem. Figure 2 shows how we include this layer into current cloud storage solutions. If an accountable node cannot adhere to the stated DHRs, the data is stored at a different node known as the destination node. The responsible node keeps a reference to the target for certain data in order to facilitate data lookup. To actualize PRADA, we add three new elements (capability, relay, and target storage), as indicated in Figure 2. Ability repository: For the purpose of locating nodes that can adhere to a certain DHR, the global capability store is consulted. Here, the cloud storage system operator specifies the DHR attributes each node in the cluster is capable of meeting. Each node maintains a local copy of the entire capability store to expedite lookups in the capability store. Due to the relative scarcity of data on DHRs and the feasibility of this technique, comprises primarily of static data. Distributing this data can be accomplished in one of two ways, depending on the specific cloud storage system: either by preconfiguring the capability store for a storage cluster, or by using the storage system to build a globally replicated view of node capabilities. We take into account all DHRs that identify the fixed characteristics of a storage node, which might be as basic as the location of the storage or as complex as the ability to delete data on a certain date. Each node manages a local relay store that contains references to information kept at other nodes. More specifically, it contains references to data that the node is in charge of but that does not adhere to the DHRs. The relay store includes one relay for each data item. the data's key, a pointer to the node where it is kept, and a copy of the DHRs. Each node has a target store where it stores data that is forwarded to it. The target store functions precisely like a conventional data store while enabling a node to discriminate between data that is covered by DHRs and data that is not.

Scalable key-value based cloud storage solutions are unlikely to be suitable for alternatives to adding an indirection layer. Although it is possible to encrypt very short DHRs in the key used for data access [23], this interferes with load balancing and necessitates knowledge of a data item's DHRs in order to compute the key for accessing it. As an alternative, replicating all relay data on all Nodes in a cluster can derive local information relay from other nodes. However, this drastically limits the cloud storage system's ability to scale and reduces the total amount of storage to the constrained space available on a single node.

We must modify storage processes (such adding and updating data) and reevaluate replication, load balancing, and failure recovery procedures in the presence of DHRs in order to integrate PRADA into a cloud storage system. We explain how we deal with these challenges in the paragraphs that follow.

4. CLOUD STORAGE OPERATIONS

The CRUD (create, read, update, delete) activities are the most significant PRADA enhancements. Here, we'll go over how we incorporate PRADA into our cloud storage model's CRUD operations (cf. Section 2.1). We presum that the coordinator node, one of the cluster's nodes, processes queries on behalf of the client (as common in cloud storage systems [26]). Clients use the capability store to choose a coordinator that complies with the desired DHRs from among the cluster's available coordinator nodes. Clients choose a coordinator based on performance measures, such as proximity, if no DHRs need to be taken into account. We defer the consideration of the effects of various replication factors for clarity reasons. and load balancing decisions to Section 5 and 6, respectively.

Create: The coordinator first determines whether DHRs are attached to a create request. The coordinator creates data according to the cloud storage system's normal procedure if no specifications are given in order to preserve the efficiency of native create requests. A create request is processed in three steps for all data with DHRs, as shown in Figure 3. Using the capability store (introduced in Section 3) to find nodes that satisfy all requested DHRs, the coordinator derives the set of eligible nodes from the received DHRs in Step 1. Customers can mix various DHR types (e.g., location and support for deletion).

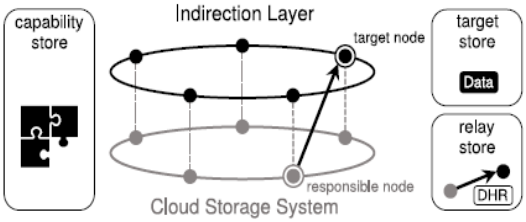

Fig. 2. System overview. PRADA adds an indirection layer to support DHRs. The capability store records which nodes support which DHR, the relay store contains references to indirected data, and the target store saves indirected data.

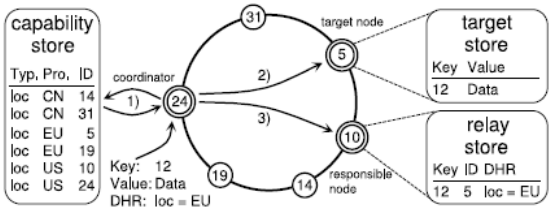

Fig. 3. Creating data. The coordinator derives nodes that comply with the DHRs from the capability store. It then stores the data at the target node and a reference to the data at the responsible node.

Nodes are regarded as qualified if they support at least one of the above characteristics for each requested type (for example, one out of numerous locations that are acceptable). Now that the coordinator is aware of which cluster nodes may meet all of the user-specified conditions, he or she must select the target node from the list of candidates on which to store the data. It's crucial to choose the target such that the cluster's overall storage burden stays balanced (we defer the discussion of this issue to Section 6). Step 2 of the process involves the coordinator sending the data to the target, who then saves it in its target store. In order to enable finding data upon read, update, and delete requests, the coordinator advises the responsible node to save a reference to the actual storage location of the data in Step 3. The coordinator applauds the productive insertion comes after the proper completion of all three processes. Although logically distinct, the second and third steps are carried out in parallel to speed up create operations.

Read: In PRADA, read requests are processed in three phases, as shown in Figure 4. The coordinator launches a typical read query at the accountable node in Step 1 using the key provided in the request. If the responsible node doesn't store the data locally, it looks for a reference to another node in its local relay store. If it does, the responsible node passes the read request to the target mentioned in the reference in Step 2 along with instructions on how to contact the coordinator node. The target seeks up the required data in Step 3 and gives the coordinator a straight response to the inquiry. Upon learning the target's outcome, The coordinator treats the outcomes the same way she would any other query outcome. If the responsible node maintains the requested data locally (for instance, if it wasn't stored with DHRs), it immediately responds to the request with the cloud storage system's standard approach. In contrast, PRADA will claim that no data was found using the typical process of the cloud storage system if the responsible node neither physically stores the data nor makes a reference to it.

Update: The updating of previously saved data entails both a (perhaps partial) updating of previously recorded data as well as a potential updating of related DHRs. In the context of this study, we stipulate that DHRs included in an update request take precedence over those included in a creation request and earlier updates. By significantly modifying the PRADA update mechanism, it is possible to provide additional semantics, such as merging old and new DHRs. So, we handle update requests in the same manner as create requests (as it is often done in cloud storage systems). The responsible node must update its relay store whenever an update request needs to alter the target node of stored data (due to modifications in the DHRs supplied). Additionally, the data must be updated according to the request (At the moment, it is kept at the old target node). The responsible node then gives the old target node the order to transfer the data to the new target node. The updated data is applied to the data by the new target node, who then acknowledges the successful update to the coordinator and responsible node and updates the relay information after locally storing the outcome. The performance of native requests is unaffected in comparison to a system that has not been updated because updates for data without DHRs are sent directly to the accountable node. Delete. Similar to how read requests are handled, delete requests are forwarded to the responsible node for the key that needs to be.

Deleted: If the responsible node saves the data on its own, it deletes the data as in a system that has not been updated. If it only keeps a reference to the data, on the other hand, it deletes the reference and sends the target an erase request. The target removes the data and notifies the coordinator that the deletion was successful. We leave to Section 7 a discussion of recovering from delete failures.

5. REPLICATION

Replication is used by cloud storage systems to achieve high availability and data durability [26]: A data item is kept on r nodes instead of only one (usually with a replication ratio of 1 r 3). The r nodes in key-value based storage systems are selected depending on the data's key (see Section 3). We cannot employ the same replication approach when adhering to DHRs.

Thus, we explain in the sections that follow how PRADA actually does replication.
Gathering Data: The coordinator chooses r targets from among the eligible nodes rather than just one. According to the cloud's replication plan, the coordinator delivers the data to all r targets and the list of all r targets to the r responsible nodes storage device). Each of the r responsible nodes is aware of all r targets as a result, and it is able to update its relay store accordingly.

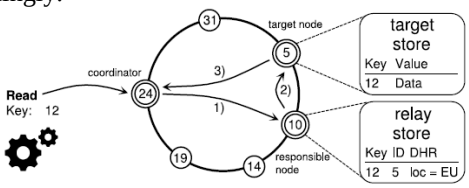

Fig.4.Reading Data: The coordinator contacts the responsible node to fetch the data. As the data was created with DHRs, the responsible node forwards the query to the target, which directly sends the response back to the coordinator.

Reading Data: The coordinator sends the read request to all accountable nodes for processing. In order to transfer a read request to one of the r target nodes, a responsible node that receives one for data that it does not locally hold searches for the targets in its relay store. Each responsible node employs the same consistent mapping between responsible and target nodes, which is calculated based on node identifiers, to guarantee that each target node receives a request. Every target that receives a read request transmits the requested information to the request coordinator. To boost dependability, each responsible node will send the read query to all r target nodes if it is reissued as a result of a failure (see Section 7). To increase reliability, each responsible node will send the request to all r target nodes.

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.

6. LOAD BALANCING

By spreading stored data and read requests evenly among the nodes, load balancing in cloud storage systems tries to reduce (long-term) load discrepancies in the storage cluster. Existing load balancing strategies need to be redesigned since PRADA fundamentally alters how data is assigned to and retrieved from nodes. Following a formal metric for measuring load balance, we will outline how PRADA creates a storage cluster that is load-balanced.

Load balance metric: An effective load balancing, or the reduction of the imbalance between the load on the nodes, seeks to load each node (almost) equally. Overloaded nodes significantly reduce the overall performance of the cloud storage system, whereas underloaded nodes squander resources. The global standard deviation of the load is normalised with the mean load across all nodes to get the load balance of a cloud storage system [48]:

$$\mathfrak{L} := \frac{1}{\mu}\sqrt{\frac{\sum_{i=1}^{|N|}(\mathfrak{L}_i - \mu)^2}{|N|}}$$

with Li representing the node I N's load. We have to minimise L in order to achieve load balance. Following the logic of a healthy load balance, this statistic particularly penalises outliers with exceptionally low or high loads.

Load Balancing in PRADA:A satisfactorily balanced load is achieved by key-value based cloud storage systems in two steps: I equal data distribution at insert time, for example by applying a hash function to identifier keys, and (ii) re-balancing the cluster if absolutely necessary by shifting data between nodes. Additional methods are supported by more sophisticated systems, such as load balancing across geographical regions [28]. We concentrate on the characteristics of key-value based storage because our goal in this research is to demonstrate the general viability of providing data compliance in cloud storage.
PRADA can handle relocating data in the event of node failures and rebalancing a cluster by shifting data between nodes (Section 7). So, in the sections that follow, we concentrate on the difficulty of load balancing in PRADA at insert time. As load balancing for indirection information is accomplished with the conventional procedures of key-value based cloud storage systems, such as by hashing identification keys, here we concentrate on equal distribution of data with DHRs to target nodes.
Compared to cloud storage systems based on keys and values, load balancing in PRADA is more difficult: The eligible target nodes are not always equal while processing a create request since they can be able to adhere to various DHRs. As a result, certain eligible nodes might provide needs that are frequently asked for but rarely supported. Future demand forecasting is notoriously challenging [49], hence we advise basing the load balancing choice on the nodes' current workload. To accomplish this, each node in the cluster must be aware of the load placed on the other nodes. The majority of cloud storage systems already exchange this information or are capable of doing so, for example via effective gossiping protocols [50]. The following is how we use this load information in PRADA. choosing the target If any of the responsible nodes are also eligible to become target nodes, PRADA first determines which of those are eligible and chooses those as target nodes first. Due to the fact that we eliminate the indirection layer in this situation, we are able to improve the performance of CRUD requests. PRADA chooses the target nodes with the lowest load

among the remaining nodes. Each node in PRADA keeps track of any creation requests it is associated with so that it can access more timely load information. A node's temporary load information is updated whenever it stores fresh data on its own or passes data to other nodes for storage. In order to bridge the gap between two updates of the load information, this temporary node information is used This strategy enables PRADA to quickly create a (nearly) equally balanced storage cluster and adapt to various usage scenarios, as we will demonstrate in Section 9.2.

7. FAILURE RECOVERY

We must take care to avoid interfering with cloud storage systems' failure recovery methods when adding support for DHRs. With PRADA, we must explicitly deal with dangling references—that is, references pointing to nodes that do not really contain the associated data—and unreferenced data—that is, data that is stored on a target node but for which there is not yet a reference that points to it. These discrepancies may be caused by errors in the (updated) CRUD operations as well as from DHR-instigated actions, such as deletions that necessitate propagating meta data to the appropriate responsible nodes.

Create: Create requests must instruct the responsible node to store the reference while also transmitting data to the target node. The coordinator can identify failures during these activities by looking for missing acknowledgments. These problems need to be rectified by rolling back and/or reissuing actions, such as changing the reference and choosing a new target node. Even so, a coordinator's own failure during the procedure could result in data that is inaccessible. We advise against implementing comparable consistency checks directly into creation operations because such errors occur seldom [51]. Instead, the lack of acknowledgements allows the client to identify the coordinator's faults. In this situation, the client notifies all applicable nodes to remove the unreferenced data and then issues the creation operation once more using a different coordinator.

Read. As opposed to the A read request doesn't alter any state in the cloud storage system like certain other activities do. Since no missing acknowledgment indicates a failure, read requests are simply reissued; no additional error handling is necessary.

Update. Despite the complexity of update operations being higher than create operations, failure management is equivalent in both cases. The storage state is assured to stay consistent since the responsible node updates its reference only after receiving the acknowledgment from the new target node. Therefore, if mistakes happen, the coordinator can rerun the operation using the same or a different destination node and carry out the necessary clean-ups. On the other hand, information regarding the potential new target node is lost if the coordinator fails. The client fixes this mistake in a manner similar to create operations by notifying the failure to all eligible nodes. The appropriate nodes then start a clean-up to guarantee a constant storage state. Delete. A responsible node may delete a reference while deleting data, but it can forget to alert the target node to also delete the reference. The lack of the appropriate acknowledgment makes this issue simple to spot for the coordinator and client. The coordinator or client will then send out a broadcast message to the target node to remove the matching data item. Given that consistency tests for delete operations are rarely unsuccessful, this strategy is more rational than doing so directly [51].

action propagation to the target node. Clients initiate CRUD operations. Data relocation or deletion, however, can potentially result in hanging references or unreferenced data. caused by the storage cluster or by DHRs that, for instance, specify a maximum data lifetime. Storage nodes also undertake data deletion and relocation through a coordinator, i.e., they choose one of the other nodes to carry out update and delete operations on their behalf, in order to maintain the consistency of the cloud storage system. As a result, deletion and relocation tasks may be properly monitored, and repair actions can be started. The same mitigations as for CRUD operations (initiated by clients) are applicable in the event that either the starting storage node or the coordinator fails while processing a query. Storage system operators have the option of using commit logs to protect against the uncommon situations where both the starting storage node and the coordinator fail while performing an operation e.g., based on Cassandra's atomic batch log [52].

8. IMPLEMENTATION

We fully deployed PRADA on top of Cassandra for the practical evaluation of our strategy [26] (our implementation is available under the Apache License [20]). With installations of up to 75 000 nodes [53] and strong scalability even over many data centres [54], Cassandra is a distributed database that is actively used as a key-value based cloud storage system by more than 1500 enterprises. This makes it particularly appropriate for our case. In order to demonstrate the adaptability and flexibility of our design, Cassandra also implements sophisticated capabilities that go beyond straightforward key-value storage, such as column orientation and searches over ranges of keys. Cassandra divides its data into a number of logical databases referred to as key spaces. Tables that are in a critical area are column families, which have both rows and columns. Each node is aware of all other nodes and their hash table ranges. Cassandra efficiently distributes this knowledge, detects node failure, and exchanges node status, such as load information, using the gossiping protocol Scuttlebutt [50]. Our approach is conceptually compatible with newer versions of Cassandra, but our implementation is based on version 2.0.5. data repositories. Three information stores are used by PRADA: the global capability store, the relay

store, and the target store (cf. Section 3). As explained in the section below, we implement each of these as a distinct key space in Cassandra. Our first realisation is that each node keeps a complete copy of the global capability store, which is a crucial space that is globally replicated across all nodes. initialised at the same time as the cluster to optimise the performance of create operations. For each DHR type, we establish a column family on this key space (as introduced in Section 2.2). A node adds all DHR properties it supports for each DHR type (as locally set by the cloud storage system operator) into the associated column family when it enters the cluster. The replication strategy of the appropriate key space then automatically replicates this information to all other nodes in the cluster. We also establish a similar relay store and target store as key spaces for each database's regular key space. The replication factor and replication mechanism in this case are inherited by the relay store from the corresponding The relay store will be replicated in the same way as the standard key store in order to achieve replication for PRADA as described in Section 5. As a result, we build a column family in the relay key space that serves as the relay store for each column family in the related key space. By creating a key space for each key space that corresponds to the key space where actual data will be stored, we take a similar approach to implementing the target store. We make an exact replica of each column family from the original key space in the target key space, which will serve as the target store. However, we develop a DHR-agnostic replication method for the target store to make sure that DHRs are followed utilise the relay storage for data addressing.

Relay and target stores must be generated every time a new key space or column family is added, whereas the global capability store is produced when the cluster is started. As a result, we initialise the necessary relay and target stores by hooking into Cassandra's Create Keyspace Statement class to detect requests for generating key spaces and column families. building load balancing and data. We support the definition of arbitrary DHRs in text form for INSERT requests so that clients can specify them while entering or modifying data (cf. Section 2.1). In order to achieve this, we expand the grammar from which the CQL3 lexer and parser derive by adding an optional postfix WITH REQUIREMENTS to INSERT statements. ANTLR is used to create Cassandra's SQL-like query language [56]. Arbitrary DHRs can be supplied using the WITH REQUIREMENTS statement and separated by the keyword AND, for example, INSERT... WITH REQUIREMENTS location = "DE," "FR," "UK," AND encryption = "AES-256." In this case, the inserted data can be stored on any node in Germany, France, or the United Kingdom that supports AES-256 encryption. Users can define any DHRs covered by our formalised model of DHRs using this method (cf. Section 2.2).

We improve Cassandra's Query Processor, notably its getStatement method for processing INSERT requests, to recognise and handle DHRs in create requests (cf. Section 4). We base our choice of acceptable nodes for processing requests with DHRs (provided using the WITH REQUIREMENTS line) on the global capability store. If nodes offer at least one of the above qualities for each desired type, they are qualified to store data with a particular set of DHRs (e.g., one out of multiple permitted locations). In order to speed up reads for the relevant key later on, we give priority to nodes that Cassandra would choose without DHRs. Otherwise, we chose nodes based on our load balancer (cf. Section 6). We perform load balancing using Cassandra's gossiping mechanism [26], which keeps a map of all nodes and their accompanying loads. We gain access to this data using Cassandra's getLoadInfo method and add local load estimators to the load information. We update the associated local estimator with the data whenever a node submits a create request or stores data on its own the inserted data's size. In order to achieve this, we hook into the methods in Cassandra's Modification Statement, Row Mutation Verb Handler, and Storage Proxy classes as well as our methods for handling requests with DHRs. These methods are invoked when data is modified locally or transmitted to other nodes. analysing data We alter Cassandra's ReadVerbHandler class to execute read requests at the accountable node in order to enable reading of redirected data as specified in Section 4. This handler is used to determine whether the present node has a reference to a different target node for the requested key by locally checking the corresponding column family in the relay store. It is called whenever a node receives a read request from the coordinator. If no If there is no reference, a regular read operation is performed by the node. Otherwise, the node uses Cassandra's sendOneWay method to submit a modified read request to one deterministically chosen target node (cf. Section 5), requesting the data on behalf of the coordinator. Target nodes then deliver the information directly to the coordination node (whose identifier is included in the request). We also add comparable tests to the LocalReadRunnable subclass of StorageProxy to correctly resolve references to data where the coordinator of a query is also the responsible node.

9. EVALUATION

We run benchmarks to measure query execution times, storage space used, and traffic used. We also simulate the load behaviour of PRADA and analyse its applicability in three real-world use cases.

9.1 Benchmarks

We first benchmark query execution time, storage space used, and bandwidth used. We evaluate PRADA's performance in every scenario in comparison to that of a Cassandra installation that hasn't been altered and PRADA*, a system that runs PRADA but only receives

raw data without DHRs. This allows us to confirm that PRADA does not actually affect data without DHRs.
Ten nodes were assembled into a cluster, which was connected via a gigabit Ethernet switch. Each node has an Intel Core 2 Q9400, 4 GB of RAM, and either 160 GB or 500 GB of storage, and it runs either Ubuntu 14.04 or 16.04 software. To prevent potential bias brought on by employing only one particular type of artificial DHR, we provide each node a unique artificial DHR attribute, (such as a space for storage). Clients uniformly choose exactly three of the available characteristics when inserting or editing data. Each column of data has 10 rows, each of which has 200 B of uniformly random data (plus 20 B for the key). These are relatively conservative estimates since as storage size increases, PRADA's relative overhead reduces. We ran 5 runs, each with 1000 operations, for each outcome, all in a single burst, or as soon as the coordinator could manage. The mean value for carrying out a single operation is shown below along with 99% confidence intervals. With the introduction of our solution, we now provide more details on how to carry out these measurements [20].

Query completion time: The query completion time (QCT) measures how long it takes the coordinator to complete a query, from when it is received until when it returns the response to the client. The replication factor and round-trip time (RTT) between cluster nodes both have an impact on it.

We first investigate how RTTs affect QCT for a replication factor of r = 1. To do this, we use netem [57] to simulate RTTs of 100 to 250 ms by artificially adding latency to outbound packets for inter-cluster communication. Our selection includes RTTs that have been noticed in global cloud data centre communications [58] and confirmed by testing in the Microsoft Azure cloud. The QCTs for the various CRUD operations and RTTs are shown in Figure 5. We notice two things. First, QCTs of PRADA* are indistinguishable from those of Cassandra, suggesting that PRADA does not degrade data without DHRs. Furthermore, PRADA's additional overhead ranges from 15.4 to 16.2% for create, 40.5 to 42.1% for read, 48.9 to 50.5% for update, and 44.3 to 44.8% for delete. The overheads for read, update, and delete are equal to the additional 0.5 RTT introduced by the indirection layer, with updates having a slightly higher overhead due to the need to erase data held at potentially outdated target nodes. Corner instances when the coordinator is also in charge of data storage lead to QCTs below the RTT.

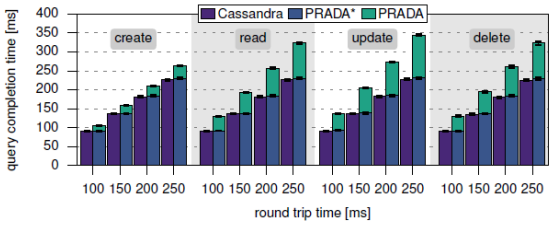

Fig. 5: Query time vs. RTT. PRADA constitutes limited overhead for operations on data with DHRs, while data without DHRs is not impacted.

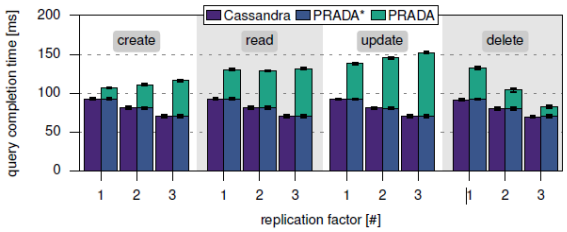

Fig. 6: Query time vs. replication. Create and update in PRADA show modest overhead for increasing replicas due to larger message sizes.

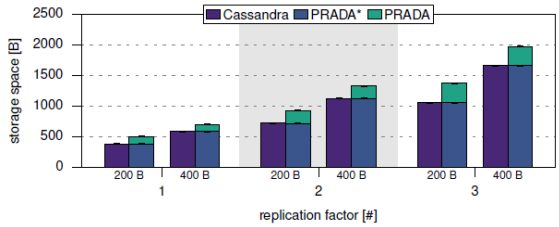

Fig. 7: Storage vs. replication. PRADA constitutes only constant over-head per DHR affected replica, while not affecting data without DHRs.

These optimizations do not apply to reads (overhead increasing from 38 to 61 ms) or updates (overhead increasing from 46 to 80 ms), as doing so would need the coordinator to be both the target and responsible node at the same time, which only occasionally occurs. Additionally, the expense of managing r references at r nodes contributes to the increase in QCTs for creates and updates, but the rise for reads is due to the additional 0.5 RTT for the indirection layer. By raising the possibility that the coordinator node is at least either the responsible or target node, the overhead for deletes falls from 41 to 12 ms for a given replication factor, preventing the need for further communication. occupied storage capacity. To determine how much extra storage space PRADA requires, Using the cfstats option of Cassandra's nodetool programme, we calculate the amount of storage space used once data has been entered. With replication factors of r = 1, 2, and 3, we perform insertions for payload sizes of 200 and 400 B (plus 20 B for the key), filling 10 columns with 20 and 40 B payload in each query. We see, for instance, a mean payload size of 312 B for an IoT data platform in real-world use cases (cf. Sec We multiply the total amount of storage space consumed by the quantity of insertions, and Figure 7 displays the average

eISSN 1303-5150

amount of storage space consumed per inserted row throughout all runs. For Cassandra to store 200 billion payload units and 585 billion for a 400 B payload. The necessary storage capacity rises by about 90% for each subsequent clone. A steady overhead of about 110 B is added by PRADA for each reproduction. The crucial finding in this case is that while the precise overhead of PRADA depends on the encoding of relay information, it does not depend on the quantity of the stored data. Even for very small payload sizes, such as a mean payload size of 133 B in a microblogging use case (cf. Section 9.3), PRADA only increases the overhead of relative storage by about 38% on top of the overhead of more than 136% already introduced by Cassandra. The storage overhead of PRADA becomes insignificant when bigger payload sizes are taken into account, such as when storing emails with a mean size of The overhead for indirection information is only 3% of the data size in 3626 B (cf. Section 9.3).

10. RELATED WORK

Our discussion of related work is divided into categories based on the several DHRs that each one addresses. Additionally, we go over ways to guarantee that DHRs are respected. distributing data storage. Data splitting amongst several storage systems is suggested in a class of related work to impose storage location requirements. In order to transparently distribute data to various cloud storage providers in accordance with DHRs, Wuchner et al. [12] and CloudFilter [18] add proxies between clients and operators. NubiSave [19] allows combining resources of various storage providers to satisfy individual redundancy or security requirements of clients. These methods can only consider individual storage systems as "black boxes." As a result, they only support a tiny portion of fine-grained DHRs within the database system itself.

Sticky policies: The idea of sticky policies offers to attach usage and obligation policies to data when it is outsourced to other parties, much to our concept of establishing DHRs [31]. Sticky policies, in contrast to our approach, focus primarily on the goal of data usage, which is mostly achieved through access restriction. Sticky policies have an intriguing property that allows them to "stick" to the related data utilising cryptographic techniques that could also be used for PRADA. Sticky policies have been developed in the context of cloud computing to define requirements on the safety and location of storage nodes [32]. It is still unclear, though, how this might be effectively implemented in a sizable, distributed storage system. We provide PRADA as a a way to make this happen. application of the law. Betg'e-Brezetz et al. [13] keep track of virtual machine access to shared file systems and only permit access if a virtual machine complies with the policy in order to enforce privacy regulations when accessing data in the cloud. To enforce data encryption, Itani et al. [14] propose using cryptographic coprocessors to create trusted and separated execution environments. Espling et al[15] .'s goal is to give service owners control over where their virtual machines are located in the cloud, enabling them to achieve specified regional deployments or offer redundancy by avoiding co-locating crucial components. These methods are unrelated to our study since PRADA focuses on the issue of assisting DHRs when processing data, whereas these methods largely concentrate on enforcing policies when processing data.

Location-based storage: The goal of data sovereignty, which Peterson et al. [16] present with a singular focus on location criteria, is to guarantee that a provider maintains data at claimed physical locations, such as those determined by measurements of network delay. Similar to this, LoSt [17] uses a challenge-response system to facilitate storage location verification. The more fundamental problem of achieving the capabilities for supporting arbitrary DHRs is the one that PRADA concentrates on.

Controlling placement of data: By arranging data mostly based on string names, SkipNet [74], which primarily focuses on distributed hash tables, offers control over data placement. Similar to how Zhou et al. [75] regulate the placement of data at a coarse grain by encoding physical topology using location-based node identifiers. Contrary to PRADA, these approaches require knowledge of the precise DHRs of the data in order to locate it because the identity of the data must be modified based on the DHRs. CRUSH [76], which focuses on distributed object-based storage systems, uses hierarchies and data distribution policies to regulate where data is placed in a cluster. These data distribution policies cannot provide the same flexibility as PRADA since they are constrained by a predetermined hierarchy. The same is true for Tenant-Defined Storage [77], which enables clients to they keep their data in DHRs. In contrast to PRADA, all of a client's data must have the same DHRs. To speed up large data analyses, SwiftAnalytics [78] suggests controlling the placement of data. In this case, the abstraction given by PRADA's method of supporting DHRs cannot be used; instead, data can only be directly placed on certain nodes.

Hippocratic Databases: Hippocratic databases keep data and a purpose description together [79]. This enables them to realise data retention after a specific amount of time and ensure the intentional use of data through access control. It is possible to build an auditing system using Hippocratic databases to determine whether a database is adhering to its data disclosure regulations [33]. This notion, however, only takes into account a single database and does not take into account distributed settings where storage nodes have varying capacities for managing data. Assurance. De Oliveira et al. [80] offer an architecture to automate the monitoring of compliance to DHRs when

transferring data in order to provide assurance that storage operators adhere to DHRs. This is also possible, as demonstrated by Bacon et al. [34] and Pasquier et al. [5] via information flow control. Similar to this, Massonet et al. [41] suggest an architecture for monitoring and audit logging in which the infrastructure provider and service provider work together to guarantee data location compliance. These methods, which are unrelated to our research, could be used to confirm that administrators of cloud storage systems use PRADA in an ethical and error-free manner.

11. RESULT

Fig 6. Admin Login Page

Fig 7. Admin Home Page

Fig 8. Operator Login Page

Fig9. Files Page

Fig 10. Customer Registration Page

12. CONCLUSION AND FUTURE ENHANCEMENT

Legislative, organisational, or customer expectations make accounting for compliance with data handling requirements (DHRs), i.e., providing control over where and how data is held in the cloud, increasingly crucial. Despite these incentives, there aren't many workable solutions to this problem in the current cloud storage systems. In this research, we introduced PRADA, which enables cloud storage operators to enforce a wide range of fine-grained DHRs and allows clients to select them. Our findings demonstrate that it is possible to support DHRs in cloud storage systems. Of course, the greater security and adaptability provided by DHRs has a cost: Although we maintain a nearly ideal storage load balance and achieve constant storage overhead, we see a slight increase in query completion times even in difficult situations.

But more importantly, PRADA has no negative effects on data without DHRs. A responsible node can locally verify that no DHRs are applicable to the data when it receives a request for it. The INSERT statement for create requests either contains DHRs or it does not, and this can be quickly and locally verified. In contrast, PRADA simply and locally checks whether a reference to a destination node for this data exists for read, update, and delete requests. This phase has minimal overhead, similar to that of running an if statement. PRADA only creates overhead if a reference is present, which suggests that the data was added via DHRs. Our thorough analysis confirms that PRADA displays the same query completion for data without DHRs. In all settings taken into account, the Cassandra system performed indistinguishably from PRADA* in terms of execution times, storage overhead, and bandwidth consumption (Figures 5 to 8). As a result, clients can decide (even down to individual data items) if DHRs are worth a slight performance hit.

The foundation of PRADA's design is a transparent indirection layer that efficiently manages DHR compliance. Our solution is constrained by this design choice in three ways. First, the degree to which the nodes' capacities to fulfil certain DHRs match the actual DHRs requested by the customers will determine the overall feasible load balance. However, PRADA is possible to achieve virtually ideal load balancing for a particular circumstance, as demonstrated in Figure 10. Second, indirection adds 0.5 round-trip times of

overhead for reads. updates, removes, etc. Only encoding selected DHRs in the key used to access the data [23] can further reduce this overhead, but this requires that everyone accessing the data have access to the DHRs, which is improbable. Replicating all relay data to the cluster's nodes would result in a fundamental improvement, but this approach is only practical for small cloud storage systems and does not provide scalability. Although we contend that indirection is probably impossible to avoid, we continue to raise this as a topic for future study. Third, how can clients be convinced that an operator is actually enforcing their DHRs and that there have been no mistakes in the DHR specification? Numerous studies have been done on this [16], [33], [41], [80] and PRADA can also benefit from the suggested strategies, such as audit logging, information flow management, and provable data custody.

The key-value paradigm is sufficiently general to serve as a useful starting point for storage systems that are based on other paradigms, even if we restrict our technique for ensuring data compliance in cloud storage to key-value based storage systems. Additionally, PRADA's design is adaptable enough to be used to different storage systems with a little more effort. For instance, applications can impact data locality (to improve speed) by carefully selecting keys in Google's globally distributed database Spanner (rather than a key-value store or a multi-version database) [28]. By altering Spanner's method of directory-bucketed key-value map-pings, PRADA might be used with Spanner. PRADA could also produce data.compliance for main memory distributed databases, such as VoltDB, where data tables are split horizontally into shards [29]. Here, DHRs could be taken into consideration while deciding how to distribute shards among the cluster's nodes. For commercial products that slice up data, like Clustrix [30], similar adjustments could be made.

To sum up, PRADA fixes a problem that both users of cloud storage systems and their operators have, namely the absence of support for DHRs. PRADA enables the use of cloud storage systems for a variety of clients who previously had to refrain from outsourcing storage, for example because of compliance with applicable data protection regulations, by offering the implementation of arbitrary DHRs while storing data in cloud storage systems. At the same time, we provide cloud storage providers with a useful and effective method of dealing with regulatory variations and expanding their customer base.

REFERENCES

[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.

eISSN 1303-5150

[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.
[51] J. K. Nidzwetzki and R. H. G¨ uting, "Distributed SECONDO: A Highly Available and Scalable System for Spatial Data Processing," in SSTD, 2015.
[52] DataStax, Inc., "Apache CassandraTM 2.0 Documentation," http: //docs.datastax.com/en/cassandra/2.0/pdf/cassandra20.pdf, 2016, last updated: 21 January 2016.
[53] The Apache Software Foundation, "Apache Cassandra," https:// cassandra.apache.org/.
[54] T. Rabl et al., "Solving Big Data Challenges for Enterprise Application Performance Management," Proc. VLDB Endow., vol. 5, no. 12, 2012.
[55] The Apache Software Foundation, "Cassandra Query Language (CQL) v3.3.1," https://cassandra.apache.org/doc/cql3/CQL.html, 2015.
[56] T. J. Parr and R. W. Quong, "ANTLR: A predicated-LL(k) parser generator," Software: Practice and Experience, vol. 25, no. 7, 1995. [57] S. Hemminger, "Network Emulation with NetEm," in linux.conf.au, 2005.
[58] S. Sanghrajka, N. Mahajan, and R. Sion, "Cloud Performance Benchmark Series: Network Performance – Amazon EC2," Cloud Commons Online, 2011.
[59] J. Walker, "HotBits: Genuine Random Numbers," http://www. fourmilab.ch/hotbits.
[60] IBM Corporation, "IBM ILOG CPLEX Optimization Studio," http: //www.ibm.com/software/products/en/ibmilogcpleoptistud/. [61] Dropbox Inc., "400 million strong,"

https://blogs.dropbox.com/ dropbox/2015/06/400-million-users/, 2015.
[62] AmazonWeb Services, Inc., "AmazonWeb Services General Reference Version 1.0," http://docs.aws.amazon.com/general/latest/ gr/aws-general.pdf.
[63] Microsoft Corporation, "Microsoft Azure Cloud Computing Platform & Services," https://azure.microsoft.com/.
[64] "Twissandra," http://twissandra.com/.
[65] J. Yang and J. Leskovec, "Patterns of Temporal Variation in Online Media," in Proceedings of the Fourth ACM International Conference on Web Search and Data Mining (WSDM). ACM, 2011.
[66] K. Giannakouris and M. Smihily, "Cloud computing – statistics on the use by enterprises," Eurostat Statistics Explained, 2014.
[67] The Apache Software Foundation, "Apache James Project," http: //james.apache.org/.
[68] "ElasticInbox – Scalable Email Store for the Cloud," http://www. elasticinbox.com/.
[69] B. Klimt and Y. Yang, "Introducing the Enron Corpus," in First Conference on Email and Anti-Spam (CEAS), 2004.
[70] M. Henze et al., "A Comprehensive Approach to Privacy in the Cloud-based Internet of Things," FGCS, 2016.
[71] M. Henze et al., "CPPL: Compact Privacy Policy Language," in ACM WPES, 2016.
[72] T. Pasquier et al., "Data-centric access control for cloud computing," in ACM SACMAT, 2016.
[73] Bug Labs, Inc., "dweet.io – Share your thing like it ain't no thang." https://dweet.io/.
[74] N. J. A. Harvey et al., "SkipNet: A Scalable Overlay Network with Practical Locality Properties," in USENIX USITS, 2003.
[75] S. Zhou, G. R. Ganger, and P. A. Steenkiste, "Location-based Node IDs: Enabling Explicit Locality in DHTs," Carnegie Mellon University, Tech. Rep., 2003.
[76] S. A. Weil et al., "CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data," in ACM/IEEE SC, 2006.
[77] P.-J. Maenhaut et al., "A Dynamic Tenant-Defined Storage System for Efficient Resource Management in Cloud Applications," Journal of Network and Computer Applications, 2017.
[78] L. Rupprecht et al., "SwiftAnalytics: Optimizing Object Storage for Big Data Analytics," in IEEE IC2E, 2017.
[79] R. Agrawal et al., "Hippocratic Databases," in Proceedings of the 28th International Conference on Very Large Data Bases (VLDB). VLDB Endowment, 2002.
[80] A. De Oliveira et al., "Monitoring Personal Data Transfers in the Cloud," in IEEE CloudCom, 2013.
Martin.