Practical Data Compliance for Cloud Storage

Martin Henze, Roman Matzutt, Jens Hiller, Erik Mühmer,
Jan Henrik Ziegeldorf, Johannes van der Giet, Klaus Wehrle
Communication and Distributed Systems, RWTH Aachen University, Germany
{henze, matzutt, hiller, muehmer, ziegeldorf, giet, wehrle}@comsys.rwth-aachen.de

***Abstract*—Despite their increasing proliferation and technical variety, existing cloud storage technologies by design lack support for enforcing compliance with regulatory, organizational, or contractual data handling requirements. However, with legislation responding to rising privacy concerns, this becomes a crucial technical capability for cloud storage systems. In this paper, we introduce PRADA, a practical approach to enforce data compliance in key-value based cloud storage systems. To this end, PRADA introduces a transparent data handling layer which enables clients to specify data handling requirements and provides operators with the technical means to adhere to them. The evaluation of our prototype shows that the modest overheads for supporting data handling requirements in cloud storage systems are practical for real-world deployments.**

I. INTRODUCTION

Most of today's web services outsource data to cloud storage. Frequently, customers and lawmakers insist that storage providers comply with different data handling requirements (DHRs), ranging from restricted storage locations or durations [1]–[3] to certain properties of the storage medium such as full disk encryption [4], [5]. These requirements are becoming increasingly diverse, detailed, and difficult to check and enforce [6]. At the same time, cloud storage systems are becoming more versatile, spanning different continents [7] or infrastructures [8], and even different second-level providers [9], [10]. In such systems, the decision where to store data is primarily taken with the goal to optimize reliability, availability, and performance, ignoring the demand for support of DHRs.

This apparent lack of control is not merely an academic problem. The Intel IT Center surveys [11] among 800 IT professionals that 78% of their organizations have to comply with regulatory mandates. Again, 78% of these organizations are concerned that cloud offers are unable to meet their requirements. In consequence, 57% of these organizations actually refrain from outsourcing regulated data to the cloud. The lacking control over the treatment of data in cloud storage hence scares away a large set of clients. This especially holds for the healthcare, financial, and government sectors [11].

Supporting powerful DHRs enables these clients to dictate adequate treatment of their data and thus allows cloud storage operators to break into new markets. Additionally, it empowers operators to efficiently handle differences in regulations (e.g., w.r.t. data protection and privacy) [12]. Although the demand for DHRs is widely acknowledged, practical support is still severely limited [11], [13], [14]. Related work primarily focuses on enforcing DHRs while processing data [15]–[17], limits itself to location requirements [18], [19], or treats the storage system as a black box and tries to enforce DHRs at a coarse granularity from the outside [14], [20], [21]. Practical solutions for enforcing arbitrary DHRs when storing data in cloud storage systems are still missing—a situation that is disadvantageous to both the clients and operators of cloud storage systems.

Our contributions. We propose 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 for the flexible and efficient routing of data to the storage nodes according to the imposed DHRs. In detail, we make the following contributions:

1) 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.
2) PRADA's design is *incremental*, i.e., it does not impair data without DHRs. PRADA supports any DHRs that can be expressed as properties of storage nodes and any arbitrary DHR combinations, covering a wide range of use cases.
3) We prove the feasibility of our approach by implementing it based on the distributed database Cassandra and quantifying the costs of supporting DHRs in cloud storage systems.

II. SCENARIO

With the increasing demand for sharing data and storing it at external parties [22], obeying to DHRs becomes a crucial challenge for cloud storage systems [13], [14], [23].

We consider a cloud storage system that is realized over a set of diverse nodes that are spread over different data centers [24]. To explain our approach in a simple yet general setting, we assume that data is addressed by a key (a unique identifier for each data item). Key-value based cloud storage systems [25]–[28] provide a general starting point for our line of research, since they are widely used and their underlying principles have been adopted in other cloud storage systems [29]–[31].

In such a system, DHRs enable clients (end users and companies) to stay in control over the treatment of their data, even if it is outsourced to the cloud [13], [23], [32]. To this end, a client attaches DHRs to each piece of data before it is sent to the cloud storage system. DHRs are binding for all systems involved in handling the data, i.e., data is only allowed to be stored at nodes in the cloud storage system that fulfill the DHRs imposed by the client.

DHRs constrain the storage, processing, distribution, and deletion of data in cloud storage. These constraints follow from legal (laws and regulations) [33], [34], contractual (standards

and specifications) [35], or intrinsic requirements (user's or company's individual privacy requirements) [36]–[38]. For businesses, compliance with legal and contractual obligations is crucial to avoid serious (financial) consequences [39].

Notably, in our setting compliance with DHRs is achieved and enforced by the operator of the cloud storage system. Only the operator knows about the characteristics of the storage nodes and only the operator can thus take the ultimate decision on which node to store a specific data item. Different works exist that propose cryptographic guarantees [16], [40], [41], accountability mechanisms [42], transparency [43], information flow control [6], [44], or even virtual proofs of physical reality [45] to relax trust assumptions. 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 cloud operator is not trusted.

III. Data Compliance in Cloud Storage

We introduce PRADA, our approach to support DHRs in key-value based cloud storage systems. The problem that 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 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.

A. System Overview

The core idea of PRADA is to add an indirection layer on top of a cloud storage system. Whenever a responsible node cannot comply with the stated DHRs, we store the data item at a different node, called *target node*. To enable lookup of data, the responsible node stores a reference to the target for this data item. We introduce three new storage components (capability, relay, and target store), as described in the following.

Capability store: The global *capability store* is used to look up nodes that can comply with a specific DHR. PRADA covers all DHRs that describe properties of a storage node, ranging from rather simplistic properties such as storage location to more advanced capabilities such as the support for deleting data at a specified point in time. To speed up lookups in the capability store, each node keeps a local copy, which requires the capability store to be kept consistent between nodes. This can be realized by preconfiguring the capability store for a storage cluster or by utilizing the storage system itself for creating a globally replicated view of nodes' capabilities.

Relay store: Each node operates a local *relay store* containing references to data stored at other nodes. For each data item the node is responsible for but does not comply with the DHRs posed at insertion, the relay store contains the key of the data, the node the data is actually stored at, and a copy of DHRs.

Target store: Each node stores data that is redirected to it in a *target store* which allows a node to distinguish data that falls under DHRs from data that does not.

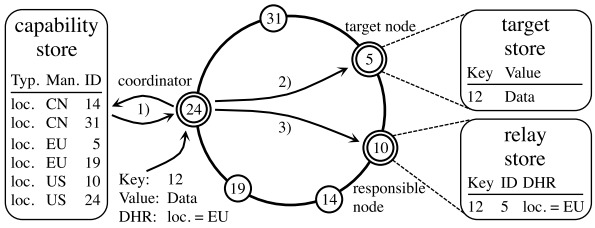

Fig. 1. **Creating data.** The coordinator derives nodes that comply with the DHRs from the capability store. It then stores the data at the target node and a reference to the data at the responsible node.

Integrating PRADA into a cloud storage system requires to adapt the individual operations such as creating and updating data and to reconsider replication and load balancing strategies.

B. CRUD Operations

The most important modifications and considerations of PRADA involve the CRUD (create, read, update, delete) operations of cloud storage systems. In the following, we describe how we integrate PRADA into the CRUD operations of our cloud storage model (cf. Section II). To this end, we assume that queries are processed on behalf of the client by one node in the cluster, the *coordinator* node. Each node can coordinate a query and clients will select them randomly.

Create. The coordinator first checks whether a create request is accompanied by DHRs. If no requirements are specified, the coordinator uses the standard method of the cloud storage system to create data so that the performance of native create requests is not impaired. For all data *with* DHRs, a create request proceeds in three steps as illustrated in Figure 1. In Step 1, the coordinator derives the set of eligible nodes based on the DHRs in the capability store. Now, the coordinator has to choose the target out of this set of eligible nodes. It is important to choose the target such that the overall storage load in the cluster remains balanced (we defer this issue to Section III-D). In Step 2, the coordinator forwards the data to the target, which stores it in its target store. Finally, in Step 3, the coordinator instructs the responsible node to store a reference to the actual storage location of the data to enable retrieving data. The coordinator acknowledges the successful insertion after all three steps have been completed successfully. The second and third step are performed in parallel to improve the query completion time of create operations.

Read. Processing read requests in PRADA is performed in three steps as illustrated in Figure 2. In Step 1, the coordinator uses the key supplied in the request to initiate a standard read query at the responsible node. If the responsible node does not store the data, it checks its relay store for a reference to a different node. Should it hold such a reference, the responsible node forwards the read request to the target in Step 2. In Step 3, the target looks up the requested data in its target store and directly returns the query result to the coordinator. Upon receiving the result, the coordinator processes it as any other query result. Should the responsible node itself store the requested data (e.g., because it was stored without DHRs), it directly answers the request. If the responsible node neither stores the data nor a reference, PRADA will report that no data was found.

Update. In this paper, we assume that DHRs of update requests supersede DHRs of create requests. We thus process update requests the same way as create requests. When the responsible node receives new information for the relay store, it checks if it already stores a reference for the corresponding key, indicating an update. In case of an update and differing old and new target nodes, the relay store needs to be updated and the data stored at the old and new target node need to be merged. To this end, the responsible node instructs the old target node to move the data to the new target node. The new target node applies the update to the data, locally stores the result, and acknowledges the successful update to coordinator and responsible node. Now, the responsible node updates the relay store information.

Delete. In PRADA, delete requests are processed analogously to read requests. The delete request is sent to the responsible node for the key that should be deleted. If the responsible node itself stores the data, it deletes the data as in an unmodified cloud storage system. In contrast, if it only stores a reference to the data, it deletes the reference and forwards the delete request to the target. The target deletes the data and informs the coordinator about the successful termination of the query.

C. Replication

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 replication factor $1 < r \leq 3$). The r nodes are chosen based on the key of the data. PRADA cannot use the same replication strategy as we have to comply with the client's DHRs. In the following, we thus detail how PRADA realizes replication.

Inserting data. Instead of selecting only one target, the coordinator picks r targets out of the set of 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).

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. It then forwards the read request to all r target nodes. Finally, a target that receives a read request sends the requested data to the coordinator. In contrast to the standard behavior, a target may receive multiple forwarded read requests. We hence ignore duplicate requests.

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.

D. Load Balancing

Load balancing aims to minimize load disparities in the cluster by distributing stored data and read requests equally among the nodes. Since PRADA drastically changes how data is assigned to and retrieved from nodes, existing load balancing schemes must be rethought. In the following, we describe a formal metric to measure load balance and then explain how PRADA builds a load-balanced storage cluster.

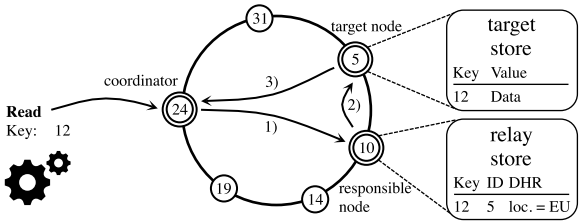

Fig. 2. **Reading data.** The coordinator contacts the responsible node to fetch the data. As the data was created with DHRs, the responsible node forwards the query to the target which directly sends the response to the coordinator.

Load balance metric. We measure the load balance of a cloud storage system by normalizing the global standard deviation of the load with the mean load μ of all nodes [46]: $\mathfrak{L} := \frac{1}{\mu}\sqrt{(\sum_{i=1}^{|N|}(\mathfrak{L}_i - \mu)^2/|N|)}$ with $\mathfrak{L}_i$ the load of node $i \in N$. To achieve load balance, we need to minimize $\mathfrak{L}$. This metric penalizes outliers with extremely low or high loads, since underloaded nodes constitute a waste of resources and overloaded nodes decrease the overall cluster performance.

Load balancing in PRADA. Key-value based cloud storage systems achieve a reasonably balanced load in two steps: (i) equal distribution of data at insert time, e.g., by applying a hash function to identifier keys, and (ii) re-balancing the cluster if absolutely necessary by moving data between nodes.

Re-balancing the cluster by moving data between nodes can be handled by PRADA similarly to the mechanism of the underlying cloud storage system. In the following, we thus focus on the challenge of load balancing during insertion.

In contrast to key-value based cloud storage systems, we face the following challenge: When processing a create request, the eligible target nodes are not necessarily equal as they might be able to comply with different DHRs. Hence, some eligible nodes might offer rarely supported but often requested requirements. Foreseeing future demands is notoriously difficult [47] and we hence take the load balancing decision based on the current load of the nodes. This requires all nodes to be aware of the load of the other nodes in the cluster. Cloud storage systems typically already exchange this information or can be extended to do so, e.g., using efficient gossiping protocols [48]. We utilize this load information in PRADA as follows. To select target nodes, PRADA first checks if any of the responsible nodes are also eligible to become a target node and selects those as target nodes. This allows us to increase the performance of CRUD requests by avoiding the indirection layer. For the remaining target nodes, PRADA selects those with the lowest load. To have access to more timely load information, each node in PRADA keeps track of all create requests it is involved with. Whenever a node itself stores new data or sends data for storage to other nodes, it increments temporary load information for the respective node. This temporary node information is used to bridge the time between two updates of the load information.

IV. EVALUATION

To thoroughly quantify and evaluate the performance of PRADA, we implemented PRADA on top of Cassandra [26].

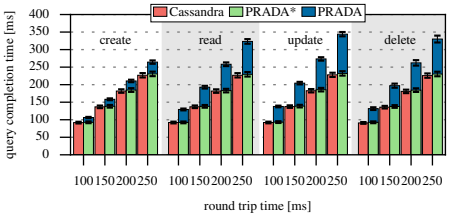

Fig. 3. **Query time vs. RTT.** PRADA constitutes limited overhead for operations on data with DHRs, while data without DHRs is not impacted.

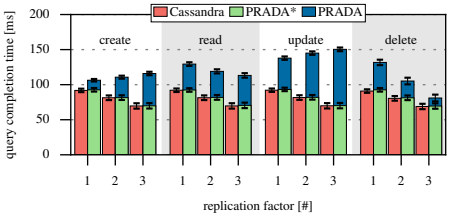

Fig. 4. **Query time vs. replication.** Create and update in PRADA show modest overhead for increasing replicas due to larger message sizes.

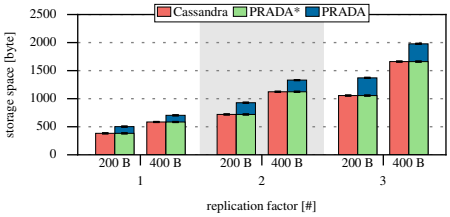

Fig. 5. **Storage vs. replication.** PRADA constitutes only constant overhead per DHR affected replica, while not affecting data without DHRs.

Cassandra is a distributed database that is employed as a key-value cloud storage system by more than 1500 companies with deployments of up to 75 000 nodes [49]. Cassandra also implements advanced features 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 PRADA. We perform benchmarks of query completion times, storage overhead, and traffic consumption. Furthermore, we study PRADA's load behavior through simulation.

A. Implementation

We implemented PRADA on top of Cassandra 2.0.5 [26], but PRADA conceptually also works with newer versions. We realize the *global capability store* as a globally replicated key space initialized at the same time as the cluster. 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 relies on Cassandra's replication mechanism, i.e., it will be replicated analogously to the regular key store. For the target store, however, we implemented a DHR-agnostic replication mechanism to ensure adherence to DHRs.

To allow clients to specify their DHRs when inserting or updating data, we modified Cassandra's method for processing requests. To this end, we add an optional postfix `WITH ANNOTATIONS` to `INSERT` statements which enables specification of multiple DHRs, e.g., `INSERT ... WITH ANNOTATIONS location = { 'DE', 'FR', 'UK' } AND encryption = { 'AES-256' }`. When processing such requests, we base our selection of eligible nodes on the global capability store, preferring nodes that Cassandra would pick without DHRs (cf. Section III-D).

Our load balancing implementation relies on Cassandra's gossiping mechanism [26] which maintains a list of all nodes together with their load. We extend this with local load change estimators, i.e., whenever the local node sends a create request or stores data itself, we update the corresponding local estimator with the size of the inserted data. Our load balancer incorporates local estimators and gossiped load information for its decision.

B. Benchmarks

We benchmark query completion time, consumed storage space, and bandwidth consumption. In all settings, we compare the performance of PRADA with the performance of an unmodified Cassandra installation and PRADA*, a system running PRADA but receiving only data without DHRs to verify that data without DHRs is not impaired by PRADA.

We set up a cluster of 10 nodes (Intel Core 2 Q9400, 4 GB RAM, 160 GB HDD, Ubuntu 14.04) interconnected via a gigabit Ethernet switch. Additionally, we use one node to interface with the cloud storage system to perform CRUD operations. We assign each node a distinct storage location. When inserting or updating data, clients request a set of three allowed storage locations uniformly randomly. Each row of data consists of 200 B (+ 20 B key), spread over 10 columns. These are conservative numbers as the relative overhead of PRADA decreases with increasing storage size. For each result, we performed 5 runs with 1000 operations and depict the mean value for one operation with 99 % confidence intervals.

Query completion time. The query completion time (QCT) denotes the time the coordinator takes for processing a query. It is influenced by the round-trip time (RTT) between nodes in the cluster and the replication factor.

We first study the influence of RTTs on QCT for a replication factor $r = 1$. To this end, we artificially add latency using `netem` [50] to emulate RTTs of 100 to 250 ms. Our choice covers RTTs observed in communication between cloud data centers around the world [51] and verified through measurements in the Microsoft Azure cloud. In Figure 3, we depict the QCTs for the different CRUD operations and RTTs. We make two observations. First, QCTs of PRADA* are indistinguishable from those of the unmodified Cassandra. Hence, data without DHRs is not impaired by our approach. Second, the additional overhead of PRADA lies between 15.9 to 16.3 % for create, 40.4 to 42.1 % for read, 49.0 to 50.7 % for update, and 43.0 to 44.6 % for delete. The overheads for read, update, and delete correspond to the additional 0.5 RTT introduced by PRADA's indirection layer and is slightly worse for updates as data stored at potentially old target nodes needs to be deleted. QCTs below the RTT result from corner cases where the coordinator is also responsible for storing data.

From now on, we fix RTTs at 100 ms and study the impact of replication factors $r = 1, 2,$ and 3 on QCTs as shown in Figure 4. Again, we observe that the QCTs of PRADA* and Cassandra are indistinguishable. For increasing replication factors, the QCTs for PRADA* and Cassandra reduce as it becomes more likely that the coordinator also stores the data. In this case, Cassandra optimizes queries. When considering the overhead of PRADA, we witness that the QCTs for creates (overhead increasing from 14 to 46 ms) and updates (overhead increasing from 46 to 80 ms) cannot benefit from these optimizations, as this would require the coordinator to be responsible and target node at the same time which happens

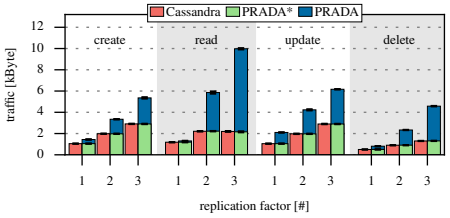

Fig. 6. **Traffic vs. replication.** Data without DHRs is not affected by PRADA. Replicas increase the traffic overhead introduced by DHRs.

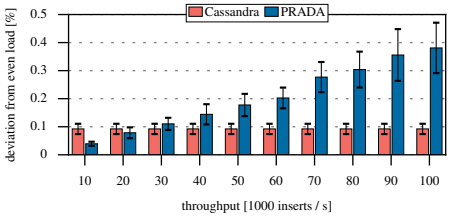

Fig. 7. **Load balance vs. throughput.** PRADA's load balance depends on insert operation throughput. Even for high throughput it stays below 0.5 %.

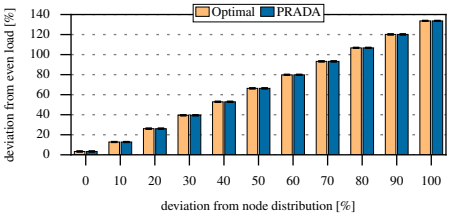

Fig. 8. **Load balance vs. distribution.** PRADA's load balance shows optimal behavior. Deviation from node distribution leads to non-even load.

only rarely. Furthermore, the increase in QCTs for creates and updates results from the overhead of handling r references at r nodes. For reads, PRADA shows an average overhead of 37 to 42 ms due to the additional 0.5 RTT for the indirection layer. For deletes, the overhead decreases from 43 ms to 18 ms for an increasing replication factor, which results from an increased likelihood that the coordinator node is at least either responsible or target node which avoids additional communication.

Consumed storage space. To quantify the additional storage space required by PRADA, we measure the consumed storage space after data has been inserted, using the `cfstats` option of Cassandra's `nodetool` utility. To this end, we conduct insertions for payload sizes of 200 B and 400 B (plus 20 B key), with replication factors of $r = 1, 2,$ and 3. We divide the total consumed storage space per run by the number of insertions and show the mean consumed storage space per inserted row over all runs in Figure 5. Each additional replica increases the required storage space by roughly 90 % for Cassandra. PRADA adds an additional constant overhead of roughly 115 B per replica. While the precise overhead of PRADA depends on the encoding of DHRs and relay information, the important observation is that it does not depend on the size of stored data. The required storage space can be further reduced by efficiently encoding DHRs [32].

Bandwidth consumption. We measure the traffic consumed by the individual CRUD operations. Figure 6 depicts the mean total generated traffic per single operation. Our results show that using PRADA comes at the cost of an overhead that scales linearly in the replication factor. When considering Cassandra and PRADA*, we observe that the consumed traffic for read operations does not increase when raising the replication factor from 2 to 3. This results from an optimization in Cassandra that requests the data only from one replica and probabilistically compares only digests of the data held by the other replicas to perform post-request consistency checks. We did not include this optimization in PRADA and hence it is possible to further reduce the bandwidth consumed by PRADA. For the other operations, the overhead introduced by PRADA ranges from 2.4 to 3.3 kB for a replication factor of 3. For a replication factor of 1, the highest overhead introduced by PRADA peaks at 1.1 kB. Thus, the traffic overhead of PRADA allows for a practical operation in cloud storage systems.

C. Load Distribution

To quantify the impact of PRADA on the load distribution of the cloud storage system, we rely on a simulation approach.

Simulation setup. As we are solely interested in the load behavior, we implemented a custom simulator in Python which models the characteristics of Cassandra with respect to network topology, data placement, and gossip behavior. On top of this simulator, we realize a PRADA cluster of n nodes, which are equally distributed among the key space [52] and insert m data items with random keys. For simplicity, we assume that all data items are of the same size. The nodes operate Cassandra's gossip protocol [48], i.e., synchronize with one random node every second and update own load information every 60 s. We repeat each measurement 10 times with different random seeds [53] and depict the mean of the load balance $\mathfrak{L}$ (cf. Section III-D) with 99 % confidence intervals.

Influence of throughput. We expect the load distribution to be influenced by the freshness of the load information as gossiped by other nodes, which correlates with the throughput of create requests. A lower throughput results in less data being inserted between two load information updates and hence the load information remains relatively fresher. To study this effect, we perform an experiment where we simulate different insertion throughputs. We simulate a cluster with 10 nodes and 10^7 create requests, each accompanied by a DHR. Even for high throughput, this produces enough data to guarantee at least one gossip round. To challenge the load balancer, we synthetically create two DHRs with two characteristics, each supported by half of the nodes such that each combination of the two DHRs is supported by two to three nodes. For each insertion we randomly select one of the possible combinations of DHRs.

Figure 7 shows the deviation from an even load for increasing throughput. We compare the results with the load distribution of a traditional Cassandra cluster. Additionally, we calculated the optimal solution under a posteriori knowledge by formulating the corresponding quadratic program for minimizing the load balance $\mathfrak{L}$ and solving it using CPLEX [54]. In all cases we observe that the resulting optimum leads to a load balance of 0, i.e., all nodes are equally loaded, and hence omit these results in the plot. Seemingly large confidence intervals result from the high resolution of our plot, showing only values below 1 %. The results show that PRADA even outperforms Cassandra for very small throughput (the load imbalance of Cassandra results from the hash function) and stays below 0.5 % even for a high throughput of 100 000 insertions/s (Dropbox processed less than 20 000 insertions/s on average in June 2015 [55]).

Influence of DHR fit. One of the core influence factors on the load distribution is the fit of clients' DHRs to the capabilities of nodes. If the distribution of DHRs in create requests heavily

deviates from the distribution of DHRs supported by the nodes, it is impossible to achieve an even load. To study this aspect, we consider a scenario where each node has a storage location and clients request exactly one of the storage locations. We simulate a cluster of 100 nodes that are geographically distributed according to the IP address ranges of Amazon Web Services [56] (US: 64 %, EU: 17 %, Asia-Pacific: 16 %, South America: 2 %, China: 1 %). First, we insert data with DHRs whose distribution exactly matches the distribution of nodes. Subsequently, we worsen the accuracy of fit by subtracting 10 to 100 % from the location with the most nodes (i.e., US) and proportionally distribute this demand to the other locations (in the extreme setting, US: 0 %, EU: 47.61 %, Asia-Pacific: 44.73 %, South America: 5.74 %, and China: 1.91 %). We simulate 10^7 insertions at a throughput of 20 000 insertions/s. To compare our results, we calculate the optimal load using a posteriori knowledge. Our results are depicted in Figure 8. We derive two insights from this experiment: i) the deviation from an even cluster load scales linearly with decreasing accuracy of fit of DHRs and node capabilities and ii) in all considered settings PRADA manages to achieve a cluster load that is extremely close to the theoretical optimum (increase < 0.03 %).

V. Related Work

To enforce storage location requirements, related work proposes to split data between different storage systems [14], [20], [21]. 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.

Similar to our idea of specifying DHRs, sticky policies have been proposed to express requirements on the security and geographical location of storage nodes [32], [57], [58]. However, it remains unclear how this could be realized efficiently in a large and distributed cloud storage system. With PRADA, we present a mechanism to achieve this goal.

To enforce privacy policies in the cloud, related work proposes monitoring access of virtual machines to data [15], trusted and isolated execution environments to enforce the encryption of data [16], and user-specified placement of virtual machines in the cloud to realize specific geographical deployments [17]. 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.

Focusing exclusively on location requirements, related work enables the verification of storage locations, e.g., based on measurements of network delay [18] or challenge-response protocols [19]. In contrast, PRADA focuses on the more fundamental challenge of supporting arbitrary DHRs.

To provide assurance that storage operators adhere to DHRs, related work suggests automated monitoring of compliance [59], information flow control [6], [44], and collaborative audit logging architectures [39]. These approaches are orthogonal to our work and could be used to verify that operators of cloud storage systems operate PRADA in an honest way.

VI. Discussion and Conclusion

Support for data handling requirements (DHRs), i.e., control over where and how data is stored in the cloud, becomes increasingly important due to legislative, organizational, or customer demands. Despite these substantial incentives, practical solutions to address this need in existing cloud storage systems are scarce. In this paper, we have proposed PRADA, which allows clients to specify a comprehensive set of fine-grained DHRs and enables cloud storage operators to enforce them accordingly. 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. Notably, data without DHRs is not impaired by PRADA. Hence, clients can choose (even for each individual data item), if DHRs are worth a modest performance decrease.

PRADA's design centers around 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 match the actual DHRs requested by the clients. However, for a given scenario, PRADA is able to nearly achieve the optimal load balance as shown in Figure 8. Second, indirection introduces an overhead of 0.5 round-trip times for reads, updates, and deletes. We believe that this indirection can not be avoided, yet this remains an open research question. Third, the indirection layer realizes functionality to support DHRs within the cloud storage system. Thus, the question arises how clients can be assured that an operator indeed enforces their DHRs. This has been widely studied [18], [39], [42], [59] and the proposed approaches such as audit logging, information flow control, and provable data possession can also be applied to PRADA (cf. Section V).

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 enabling 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., because they have to comply with 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.

Acknowledgments

The authors would like to thank Annika Seufert for her contributions to the simulation of PRADA. This work has received funding from the European Union's Horizon 2020 research and innovation program 2014-2018 under grant agreement No. 644866 (SSICLOPS) and the Excellence Initiative of the German federal and state governments. It reflects only the authors' views and the European Commission is not responsible for any use that may be made of the information it contains.

References

[1] R. Gellman, "Privacy in the Clouds: Risks to Privacy and Confidentiality from Cloud Computing," World Privacy Forum, 2009.

[2] W. Jansen, T. Grance *et al.*, "Guidelines on Security and Privacy in Public Cloud Computing," *NIST Special Publication*, vol. 800, 2011.

[3] S. Pearson and A. Benameur, "Privacy, Security and Trust Issues Arising from Cloud Computing," in *IEEE CloudCom*, 2010.

[4] United States Congress, "Gramm-Leach-Bliley Act (GLBA)," Pub.L. 106-102, 113 Stat. 1338, 1999.

[5] D. Song, E. Shi, I. Fischer, and U. Shankar, "Cloud Data Protection for the Masses," *Computer*, vol. 45, no. 1, 2012.

[6] T. F. J. M. Pasquier, J. Singh, J. Bacon, and D. Eyers, "Information Flow Audit for PaaS Clouds," in *IEEE IC2E*, 2016.

[7] V. Abramova and J. Bernardino, "NoSQL Databases: MongoDB vs Cassandra," in *C3S2E*, 2013.

[8] R. Buyya, R. Ranjan, and R. N. Calheiros, "InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services," in *ICA3PP*, 2010.

[9] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, and M. Morrow, "Blueprint for the Intercloud - Protocols and Formats for Cloud Computing Interoperability," in *ICIW*, 2009.

[10] N. Grozev and R. Buyya, "Inter-Cloud Architectures and Application Brokering: Taxonomy and Survey," *Software: Practice and Experience*, vol. 44, no. 3, 2012.

[11] Intel IT Center, "Peer Research: What's Holding Back the Cloud?" Tech. Rep., 2012.

[12] D. Catteddu and G. Hogben, "Cloud Computing – Benefits, Risks and Recommendations for Information Security," European Network and Information Security Agency (ENISA), 2009.

[13] M. Henze, R. Hummen, and K. Wehrle, "The Cloud Needs Cross-Layer Data Handling Annotations," in *IEEE S&P Workshops*, 2013.

[14] T. Wüchner, S. Müller, and R. Fischer, "Compliance-Preserving Cloud Storage Federation Based on Data-Driven Usage Control," in *IEEE CloudCom*, 2013.

[15] S. Betgé-Brezetz, G.-B. Kamga, M.-P. Dupont, and A. Guesmi, "End-to-End Privacy Policy Enforcement in Cloud Infrastructure," in *IEEE CloudNet*, 2013.

[16] 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.

[17] D. Espling, L. Larsson, W. Li, J. Tordsson, and E. Elmroth, "Modeling and Placement of Cloud Services with Internal Structure," *IEEE Transactions on Cloud Computing*, vol. 4, no. 4, 2014.

[18] 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.

[19] G. J. Watson, R. Safavi-Naini, M. Alimomeni, M. E. Locasto, and S. Narayan, "LoSt: Location Based Storage," in *ACM CCSW*, 2012.

[20] I. Papagiannis and P. Pietzuch, "CloudFilter: Practical Control of Sensitive Data Propagation to the Cloud," in *ACM CCSW*, 2012.

[21] J. Spillner, J. Müller, and A. Schill, "Creating optimal cloud storage systems," *Future Generation Computer Systems*, vol. 29, no. 4, 2013.

[22] P. Samarati and S. D. C. di Vimercati, "Data Protection in Outsourcing Scenarios: Issues and Directions," in *ACM ASIACSS*, 2010.

[23] M. Henze, M. Großfengels, M. Koprowski, and K. Wehrle, "Towards Data Handling Requirements-aware Cloud Computing," in *IEEE CloudCom*, 2013.

[24] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "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 Key-value 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, pp. 35–40, 2010.

[27] M. T. Özsu and P. Valduriez, *Principles of Distributed Database Systems*, 3rd ed. Springer, 2011.

[28] K. Grolinger, W. Higashino, A. Tiwari, and M. Capretz, "Data management in cloud environments: NoSQL and NewSQL data stores," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 2, no. 1, 2013.

[29] J. C. Corbett *et al.*, "Spanner: Google's Globally-distributed Database," in *USENIX OSDI*, 2012.

[30] M. Stonebraker and A. Weisberg, "The VoltDB Main Memory DBMS," *IEEE Data Eng. Bull.*, vol. 36, no. 2, 2013.

[31] Clustrix, Inc., "Scale-Out NewSQL Database in the Cloud," http://www.clustrix.com/.

[32] M. Henze, J. Hiller, S. Schmerling, J. H. Ziegeldorf, and K. Wehrle, "CPPL: Compact Privacy Policy Language," in *ACM WPES*, 2016.

[33] United States Congress, "Health Insurance Portability and Accountability Act of 1996 (HIPAA)," Pub.L. 104191, 110 Stat. 1936, 1996.

[34] European Parliament and the Council of the European Union, "Directive 95/46/EC on the protection of individuals with regard to the processing of personal data and on the free movement of such data," L281, 23/11/1995.

[35] PCI Security Standards Council, "Payment Card Industry (PCI) Data Security Standard – Requirements and Security Assessment Procedures, Version 3.1," 2015.

[36] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "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.

[37] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds," in *ACM CCS*, 2009.

[38] M. Henze *et al.*, "A Comprehensive Approach to Privacy in the Cloud-based Internet of Things," *FGCS*, 2016.

[39] P. Massonet *et al.*, "A Monitoring and Audit Logging Architecture for Data Location Compliance in Federated Cloud Infrastructures," in *IEEE IPDPS Workshops*, 2011.

[40] M. Henze, J. Hiller, O. Hohlfeld, and K. Wehrle, "Moving Privacy-Sensitive Services from Public Clouds to Decentralized Private Clouds," in *CLaw Workshop*, 2016.

[41] R. Hummen, M. Henze, D. Catrein, and K. Wehrle, "A cloud design for user-controlled storage and processing of sensor data," in *IEEE CloudCom*, 2012.

[42] R. Agrawal *et al.*, "Auditing Compliance with a Hippocratic Database," in *VLDB*, 2004.

[43] M. Henze *et al.*, "Towards Transparent Information on Individual Cloud Service Usage," in *IEEE CloudCom*, 2016.

[44] J. Bacon *et al.*, "Information Flow Control for Secure Cloud Computing," *IEEE Transactions on Network and Service Management*, vol. 11, no. 1, 2014.

[45] U. Rührmair *et al.*, "Virtual Proofs of Reality and their Physical Implementation," in *IEEE S&P*, 2015.

[46] A. Corradi, L. Leonardi, and F. Zambonelli, "Diffusive Load-Balancing Policies for Dynamic Applications," *IEEE Concurrency*, vol. 7, no. 1, 1999.

[47] L. Rainie and J. Anderson, "The Future of Privacy," Pew Research Center, http://www.pewinternet.org/2014/12/18/future-of-privacy/, 2014.

[48] R. van Renesse, D. Dumitriu, V. Gough, and C. Thomas, "Efficient Reconciliation and Flow Control for Anti-entropy Protocols," in *LADIS*, 2008.

[49] The Apache Software Foundation, "Apache Cassandra," https://cassandra.apache.org/.

[50] S. Hemminger, "Network Emulation with NetEm," in *linux.conf.au*, 2005.

[51] S. Sanghrajka, N. Mahajan, and R. Sion, "Cloud Performance Benchmark Series: Network Performance – Amazon EC2," Cloud Commons Online, 2011.

[52] DataStax, Inc., "Apache Cassandra™ 2.0 Documentation," http://docs.datastax.com/en/cassandra/2.0/pdf/cassandra20.pdf, 2016, last updated: 21 January 2016.

[53] J. Walker, "HotBits: Genuine Random Numbers," http://www.fourmilab.ch/hotbits.

[54] IBM Corporation, "IBM ILOG CPLEX Optimization Studio," http://www.ibm.com/software/products/en/ibmilogcpleoptistud/.

[55] Dropbox Inc., "400 million strong," June 24, 2015. [Online]. Available: https://blogs.dropbox.com/dropbox/2015/06/400-million-users/

[56] Amazon Web Services, Inc., "Amazon Web Services General Reference Version 1.0," http://docs.aws.amazon.com/general/latest/gr/aws-general.pdf.

[57] S. Pearson and M. C. Mont, "Sticky Policies: An Approach for Managing Privacy across Multiple Parties," *Computer*, vol. 44, no. 9, 2011.

[58] S. Pearson, Y. Shen, and M. Mowbray, "A Privacy Manager for Cloud Computing," in *CloudCom*, 2009.

[59] A. De Oliveira, J. Sendor, A. Garaga, and K. Jenatton, "Monitoring Personal Data Transfers in the Cloud," in *IEEE CloudCom*, 2013.