

The Case for Fast and Invariant-Preserving Geo-Replication

Valter Balegas, Sérgio Duarte,
Carla Ferreira, Rodrigo Rodrigues, Nuno Preguiça

CITI/FCT/Universidade Nova de Lisboa

Marc Shapiro
Mahsa Najafzadeh

INRIA / LIP6

Abstract—Cloud storage systems showcase a range of consistency models, from weak to strong consistency. Weakly consistent systems enable better performance, but cannot maintain strong application invariants, which strong consistency trivially supports. This paper takes the position that it is possible to both achieve fast operation and maintain application invariants. To that end, we propose the novel abstraction of *invariant-preserving CRDTs*, which are replicated objects that provide invariant-safe automatic merging of concurrent updates. The key technique behind the implementation of these CRDTs is to move replica coordination outside the critical path of operations execution, to enable low normal case latency while retaining the coordination necessary to enforce invariants. In this paper we present ongoing work, where we show different *invariant-preserving CRDTs* designs and evaluate the latency of operations using a counter that never goes negative.

I. INTRODUCTION

To improve the user experience in services that operate on a global scale, from social networks and multi-player online games to e-commerce applications, the infrastructure that supports those services often resorts to geo-replication [9], [7], [17], [18], [16], [26], [8], i.e., maintains copies of application data and logic in multiple data centers scattered across the globe, providing improved scalability and lower latency. But not always the advantages of geo-replication are exploited by worldwide services, because, when services need to maintain invariants over the data, they have to synchronize with remote data centers in order to execute some operations, which negatively impacts operations' latency. In a geo-replicated scenario, latency may amount to hundreds of milliseconds.

The impact of high latency in the user's experience is well known [22], [11] and has motivated the academia [7], [1], [9] and industry [13], [5], [25] to use weaker consistency models with low-latency operations at the trade of data consistency.

When running applications under such weak consistency models, applications in different data centers execute operations concurrently over the same set of data leading to temporary divergence between replicas and potentially counter-intuitive and undesirable user-perceived semantics.

This research is supported in part by European FP7 project (2013–2016)  Fundação para a Ciência e Tecnologia SFRH/BD/87540/2012 and PEst-OE/EEI/UI0527/2014.

Good user-perceived semantics are trivially provided by systems that use strong-consistency models, namely those that serialize all updates, and therefore preclude that two operations execute without seeing the effects of one another [8], [16]. Not all operations require strong guarantees to execute, and some systems provide both strong and weak consistency models for different operations [26], [16].

In this paper, we claim that it is possible to achieve the best of both worlds, i.e., that fast geo-replicated operations can coexist with strong application invariants without impairing the latency of operations. To this end, we propose novel abstract data types called *invariant-preserving CRDTs*. These are replicated objects that, like conventional CRDTs [23], automatically merge concurrent updates, but, in addition, they can maintain application invariants. Furthermore, we show how these CRDTs can be efficiently implemented in a geo-replicated setting by moving the replica coordination that is needed for enforcing invariants outside the critical path of operation execution.

In this paper, we discuss cloud consistency models (§II); present the concept of InvCRDT (§III), abstract data types that offer invariant-safe operations; discuss the implementation of these ADTs (§IV); discuss invariants that span multiple objects (§V-B); Present the practical benefits of InvCRDTs (§VI) and, finally, we briefly review related work (§VII) and present our conclusions (§VIII).

II. DECOMPOSING CONSISTENCY REQUIREMENTS

Recent cloud systems [8], [12], [26], [16] have adopted strong consistency models to avoid concurrency anomalies. These models rely on a serializable (or even linearizable) execution order for operations to provide the illusion that a single replica exists. They do so at the expense of lower availability on failures and increased latency for operations - a direct consequence of the CAP theorem [6], which states that there is a trade-off between availability and consistency in systems prone to partitioning.

We argue that enforcing strong consistency is not mandatory for fulfilling the requirements of most applications. We use the example of an e-commerce site to motivate such statement, by identifying three central requirements of this application.

First, users of the application must not observe a past version of any given data item after observing a more recent

one – e.g., after adding some item to her shopping cart, the user does not want to observe a shopping cart where the item is not present. A way to achieve this without per-operation replica synchronization is to support causal consistency, as found in several cloud systems [17], [18].

Second, when concurrent updates exist, data replicas cannot be allowed to diverge permanently. This requires some form of automatic reconciliation that deals with concurrent updates identically in all sites, leading to a consistency model that has been recently coined as causal+ consistency [17] or fork-join-causal consistency [19]. For example, after two users add two different items to a shopping cart, both items should be in the reconciled version of the shopping cart.

Finally, the e-commerce application has crucial integrity constraints that must be preserved despite concurrent updates – e.g., the stock of a product should be greater or equal to zero, thus avoiding that the store sells more items than what it has in stock.

In current systems, invariants as the stock example are usually preserved by running such application (or operations that can break the invariant [26], [16]) under a strong consistency model. Instead, we propose to run such applications under a consistency model that provides the following properties: causal consistency; automatic reconciliation; and invariant preservation. We call this consistency model *causal+invariants* consistency.

It seems straightforward that enforcing invariants usually requires some form of coordination among nodes of the system – e.g., to ensure that a product stock does not go negative, it is necessary that replicas coordinate so that the number of successful sales do not exceed the number of items in stock. However, unlike the solution adopted by strong consistency, in many situations this coordination can be executed outside of the critical execution path of operations. In the previous example, the rights to use the available stock can be split among the replicas, allowing a purchase to proceed without further coordination provided replica where the operation is submitted has enough rights [20], [21].

III. THE CASE FOR INVARIANT-PRESERVING CRDTs

Conflict-free replicated data-types (CRDT [23]) are data types that leverage the commutativity of operations to automatically merge concurrent updates in a sensible way. Several CRDT specifications have been proposed for some of the most commonly used data types, such as lists, sets, maps and counters, allowing rapid integration in existing applications. CRDTs provide convergence by design and, when combined with a replication protocol that delivers operations in causal order, they trivially provide causal+ consistency [17], [26].

A. The concept of InvCRDTs

In this paper, we propose the concept of invariant-preserving CRDT (InvCRDT), a conflict-free data type that maintains a given invariant even in the presence of concurrent operations – the *BoundedCounter* [under submission] implements a counter that cannot be negative.

Some CRDTs already maintain invariants internally by repairing the state – e.g., in the graph CRDT [23], when one user adds an arc between two nodes and other user concurrently removes one of the nodes, the graph CRDT does not show the arc. However, unlike these solutions, InvCRDTs maintain invariants by explicitly disallowing the execution of operations that would lead to the violation of an invariant. By having immediate feedback that an operation cannot be executed, an application can give that feedback to the users – e.g., in an e-commerce application, an order will fail if some product has no stock available, since the operation of decrementing the stock of the product, aborts when implemented with a *BoundedCounter*.

For achieving this functionality, a replica of an InvCRDT includes both the state of the object and information about the rights the replica holds. These rights allow the execution of operations that potentially break invariants without coordination while guaranteeing that the invariants will not be broken. The union of the rights granted to each of the existing replicas guarantees that the invariants defined will be preserved despite any concurrent operation. The set of initial rights will depend on the initial value of the object. For example, in a *BoundedCounter* with initial value 10 and two replicas, each replica has the rights to increment the counter at any moment and the rights to execute five decrement operations.

The rights each replica holds are consumed or extended when an operation is submitted locally – e.g., in the previous example, a decrement will consume the rights to decrement by one, and an increment will increase the local rights to decrement by one. If enough rights exist locally, it is assured that the execution of the operation in other replicas will not break the defined invariant. If not enough rights exist locally, the execution of the method aborts (in our Java-based implementation, by throwing an exception) and it has no side-effects in any replica. Optionally, when not enough rights exist locally, the system may try to obtain additional rights by transferring them from some other replica(s). In this case, the method execution blocks until the necessary communication with other replicas is done. In this case, the overhead of operation execution will tend to be similar to the overhead of providing strong consistency.

This model for InvCRDTs is general enough to allow different implementations, as discussed in the next section. An important property on InvCRDTs that must be highlighted is that InvCRDTs do not eliminate the need of coordination among replicas: they only allow the coordination to be executed outside the critical path of execution of an application request, through the exchange of rights. Next we discuss the common invariants in applications and how they can be addressed using InvCRDTs.

B. Using InvCRDTs in applications

There are many examples in the literature of applications with integrity constraints that are good candidates for using InvCRDTs.

Li et. al. [16] report that two invariants must be considered

in TPC-W. First, the stock of a product must be non-negative. This can be addressed by the *BoundedCounter* previously mentioned. Second, the system must guarantee that unique identifiers are generated in a number of situations where new data items are created. To address this requirement, the space of possible identifiers could be partitioned among the replicas (for example, using the replica identifier as a suffix). InvCRDT versions of containers (e.g., set, maps) can be created, where each replica maintains rights for assigning new unique identifiers to elements added to the object. The authors also report that similar invariants must be preserved for Rubis.

Cooper et. al. [7] discuss several applications, among them, one that maintains an hierarchical namespace. Although they do not explicitly discuss invariants, it is clear to see that there are two important invariants that should be preserved: no two objects have the same name; and no cycles exist in the presence of renames. For the first invariant, we use rights that preclude two replicas from generating identical names – a replica must acquire rights to generate identifiers with some prefix). Maintaining the second invariant is more complex and requires obtaining the exclusive right to modify the path of directories from the first common ancestor of the original and destination names for supporting renames (section V-A). This can be implemented by extending our graph CRDT [23] with these rights.

Other applications have invariants on the cardinality of containers (e.g., a meeting must have at least K members), on the properties of elements present in containers (e.g., at least one element of each gender), etc. These invariants can also be preserved by having InvCRDT versions of those containers.

More recently, Bailis et al.[2] have studied OLTP systems and summarized typical invariants that show up in applications. Some of them are instantiations of the ones described above, while other require more elaborate mechanisms as discussed in section IV.

IV. SUPPORTING INVCRDTS

We assume a typical cloud computing environment composed by clients and data centers. Data centers run application servers for handling client requests and a replicated storage system to persist application data. The effects of client requests are persisted by modifying the data stored in the system, represented as InvCRDTs. Finally, a replication protocol that delivers operations in causal order is used to achieve our proposed causal+invariants consistency model.

One possible design would consist of managing the rights associated with InvCRDTs through a centralized server. In this case, each replica would obtain these rights by contacting such central entity (as in [20], [21]). We propose an alternative approach, where the rights associated with an InvCRDT are maintained in a decentralized way, completely inside the InvCRDT.

Our generic solution consists in modelling application data as resources and by keeping the rights of each replica as a vector of (*replicaId* \Rightarrow *value*) entries for each resource type in all InvCRDT replicas. Each operation is modelled as consuming

or creating resources. For example, in the *BoundedCounter*, a single resource type exists, and a resource corresponds to one unit in the counter; an increment creates one resource; a decrement consumes one resource. In an InvCRDT that needs to generate unique identifiers, the reserved resources are a subset of the identifiers (e.g., a chunk of consecutive identifiers or a subset of identifiers ended in the reserved suffix).

Operations that modify the rights vector – consume (subtract), extend (add), transfer (atomically subtract from one entry and add to another) – are commutative. Thus, they can be supported in a convergent data-type style, where operations only need to execute in causal order in the different replicas¹. Consume and extend operations affect the rights of the replica where the operations are initiated. The transfer operation must be initiated in the replica from which the rights are to be transferred from.

This execution model guarantees that in any given replica *i*, the rights that are known to exist for replica *i* are a conservative view when considering all operations that can have been executed. The reason for this is that all operations that decrement the rights of a given replica, consume and transfer, are submitted locally, while a remote transfer that is not yet known may increase the local rights. This property guarantees the correctness of our approach.

V. DISCUSSION

A. InvCRDT data-types

In section III-A we briefly presented the design of the *BoundedCounter* CRDT. We are studying other data-types that can share the same philosophy of maintaining the state of the object as well as the rights to execute operations. The *BoundedCounter* is a fairly simple example to understand, however the same idea can be applied to other data-types.

We give the intuition for a few other data-types and what invariants they can preserve:

Tree Each node in a tree has a unique parent node. This invariant can be broken by concurrently moving a node and putting it under two different nodes. A possible solution to prevent the violation of this invariant consists in associating to each node a right to modify its subtree. When a replica acquires rights over a node it automatically acquires the rights to modify any descendent of that node. The replica that holds rights over a portion of the tree may give permission to another replica to modify some subtree, losing the permission itself to modify any node under that subtree. This strategy enforces a replica executing a rename operation to hold rights over the origin and destination names, which prevents any concurrent operation from creating a cycle.

Graph To implement a graph that is always consistent, i.e., an edge always connect to an existing node, without using the automatic convergence mechanism of the graph CRDT, we associate rights to each node, which have to be acquired in order to remove it, or connect an edge. When a new node is

¹As with CRDTs, it is possible to design an equivalent solution based on state propagation.

created it has rights associated to the replica that created the node. Preventing cycles in a graph is more complex than in trees and we have not addressed that so far.

Map Two concurrent puts in a map may end up associating different element to the same key. To prevent this situation, we can associate rights to ranges of keys which have to be acquired in order to execute a put operation. This guarantees that two different replicas cannot execute a conflicting put operation. The strategy of key domain partitioning can be used to provide unique identifiers.

We aim to provide a library of InvCRDTs that support most of the invariants that are common in applications, however we are still investigating an easy way to provide them to programmers.

B. Multi-object invariants

InvCRDTs enforce invariants in a single object. However, application invariants can often span multiple objects – e.g., a user can only checkout a shopping cart if all items are in stock.

Supporting these invariants requires enforcing some type of operation grouping. Recently, weakly consistent storage systems have provided support for some form of transactions [18], [26]. We could build on this type of support to maintain invariants over multiple objects – e.g., in the previous example, a transaction would succeed only if the data center where it was submitted holds rights to consume all the necessary stock units of each item.

Some other invariants establish relations that must be maintained among multiple objects – e.g., in a courseware application, a student can only be part of a course student group if he or she is enrolled in the course. This invariant can be maintained either by repairing (e.g., if the students enrolment in the course is cancelled, the membership in the course student group is also cancelled) or avoiding the invariant violation. It seems clear that these types of invariants can be preserved by restricting concurrent operations in multiple objects (e.g., avoiding the concurrent creation of a group and removal of a student involved). However, we are still studying the best approach to represent them as InvCRDTs. Additionally, it is also not obvious what is the best way to define invariant repairing solutions in these cases. Addressing these issues is also left as future work.

VI. PRELIMINARY EVALUATION

We conducted some preliminary experiments to evaluate the latency of InvCRDT operations. We made an Erlang prototype that extends Riak [5] with support for InvCRDTs. Basically the prototype is a middleware component that is stacked between the application server and the storage system. The middleware’s main function is to exchange rights between replicas, so that when operation are executed rights are available locally and the operation succeed without contacting any remote data center.

We implemented a micro-benchmark that simulates the manipulation of items’ stock on purchases in an e-commerce

application: Decrement operations are submitted to a counter in multiple data centers and the value of the counter cannot go negative, regardless the operations propagation frequency between data centers.

We implemented the BoundedCounter and the policies to exchange rights between replicas. These exchange of rights occur in the background and try to prevent rights from being exhausted locally. When a replica runs out of rights and executes a decrement, it tries to fetch the rights from a remote data center, which potentially has high latency.

We compare the solution using InvCRDT (BCounter) against a weak consistency (WeakC) solution that uses a convergent counter and a solution that provides strong consistency (StrongC) by executing all operations on the same data center. Riak natively support these features: the convergent counter is an implementation of the PN-Counter CRDT [23] and the strong consistency solution uses a consensus algorithm based on the Paxos algorithm [14].

We did not implemented true causality in our prototype, instead the middleware provides key-linearizability, which is sufficient because in the experiments all operations are executed in a single-object. Key-linearizability is necessary to avoid concurrent requests to use the same rights within the same data center.

A. Experimental Setup

Our experiments comprised 3 Amazon EC2 data centers distributed across the globe. We installed a Riak data store in each EC2 availability zone (US-East, US-West, EU). Each Riak cluster is composed by three m1.large machines, with 2 vCPUs, producing 4 ECU² units of computational power, and with 7.5GB of memory available. We use Riak 2.0.0pre5 version.

a) *Operations latency*: Figure 1 details these results by showing the CDF of latency for operation execution. As expected, the results show that for *StrongC*, remote clients experience high latency for operation execution. This latency is close to the RTT latency between the client and the DC holding the data. For *StrongC*, each step in the line consists mostly of operations issued in different DCs.

Both *BCounter* and *WeakC* experience very low latency. In a counter-intuitive way, the latency of *BCounter* is sometimes even better than the latency of *WeakC*. This happens because our middleware caches the counters, requiring only one access to Riak for processing an update operation when compared with two accesses in *WeakC* (one for reading the value of the counter and another for updating the value if it is positive).

Figure 2 furthers details the behaviour of our middleware, by presenting the latency of operations over time. The results show that most operations take low latency, with a few peak of high latency when a replica runs out of rights and needs to ask for additional rights from other data centers. The number of peaks is small because most of the time the pro-active

²1 ECU corresponds is a relative metric used to compare instance types in the AWS platform

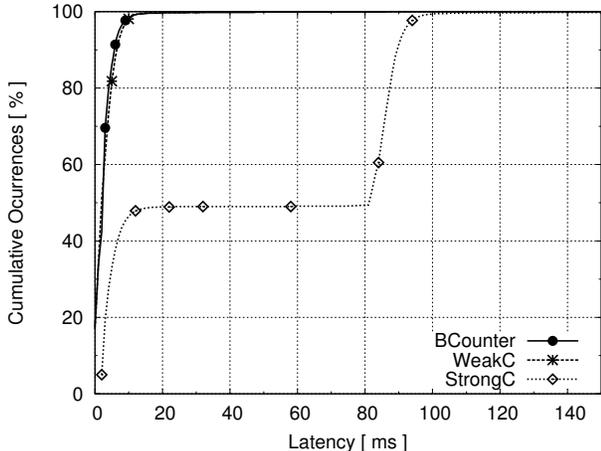


Fig. 1. CDF of latency with a single counter.

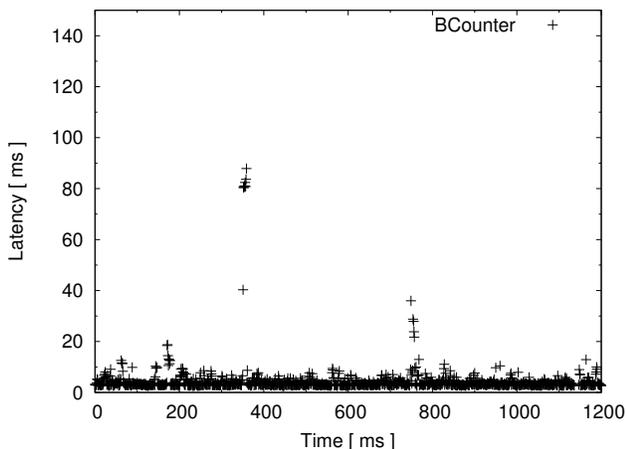


Fig. 2. Latency measured over time.

mechanism for exchanging rights is able to provision a replica with enough rights before all rights are used.

VII. RELATED WORK

A large number of cloud storage systems supporting geo-replication have been developed in recent years. Some of these systems [9], [17], [18], [1], [13] provide variants of eventual consistency, where operations return immediately after being executed in a single data center. This approach has the lowest latency possible for end-users, but since the guarantees they provide are so weak, a handful of other systems try to provide better semantics for the user and still avoid cross data center coordination, such as those that provide causal consistency [17], [1], [10], [3]. We target to provide similar ordering guarantees of messages but improve over these systems by maintaining applications invariants that require some form of coordination.

Systems that provide strong consistency[8] incur in coordination overhead that increases latency of operations. Some systems tried to combine the benefits of weak and strong consistency models by supporting both models. In Walter [26] and Gemini [16], transactions that can execute under weak consistency run fast, without needing to coordinate with other data centers.

More recently, Sieve [15] automates the decision between executing some operation in weak or strong consistency. Bailis et al. [2] have also studied when it is possible to avoid coordination in database systems, while maintaining application invariants. Our work is complimentary, by providing solutions that can be used when coordination cannot be avoided.

Escrow transactions [20] have been proposed as a mechanism for enforcing numeric invariants while allowing concurrent execution of transactions. The key idea is to enforce local invariants in each transaction that guarantee that the global invariant is not broken. The original escrow model is agnostic to the underlying storage system and in practice was mainly used to support disconnected operations [24], [21] in mobile computing environments, using a centralized solution to handle reservations.

The demarcation protocol [4] is an alternative that has been proposed to maintain invariants in distributed databases and recently applied to optimize strong-consistency protocols [12]. Although the underlying protocols are similar to escrow-based solutions, the demarcation protocol focuses on maintaining invariants across different objects.

We aim to combine these different mechanisms to provide a unified framework that programmers can use to improve the consistency of applications given the same assumptions as in weak consistency systems.

VIII. CONCLUSION

This paper presents a weak consistency model, extended with invariant preservation for supporting geo-replicated services. For supporting the causal+invariants consistency model, we propose a novel abstraction called invariant-preserving CRDTs, which are replicated objects that provide both sensible merge of concurrent updates and invariant preservation in the presence of concurrent updates. We outline the design of InvCRDTs that can be deployed on top of systems providing causal+ consistency only. Our approach provides low latency for most operations by moving the necessary coordination among nodes outside of the critical path of operation execution.

The next steps in our work are to build a library of CRDTs that programmers can use to maintain application invariants as well as providing a programming model that eases the use of these data-types in applications. One possibility would be to categorize invariants and have specific data-types to preserve each of them with low-latency. We are also still studying how to maintain invariants that span multiple objects and what guarantees the replication model must provide in order to maintain them.

The preliminary evaluation showed that it is possible to

maintain invariants under weak consistency by relying on a proactive rights exchange mechanism to transfer rights between replicas.

REFERENCES

- [1] S. Almeida, J. a. Leitão, and L. Rodrigues. Chainreaction: A causal+consistent datastore based on chain replication. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 85–98, New York, NY, USA, 2013. ACM.
- [2] P. Bailis, A. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Coordination-avoiding database systems. *CoRR*, abs/1402.2237, 2014.
- [3] P. Bailis, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Bolt-on causal consistency. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 761–772, New York, NY, USA, 2013. ACM.
- [4] D. Barbará-Millá and H. Garcia-Molina. The demarcation protocol: A technique for maintaining constraints in distributed database systems. *The VLDB Journal*, 3(3):325–353, July 1994.
- [5] Basho. Riak. <http://basho.com/riak/>, 2014. Accessed Jan/2014.
- [6] E. A. Brewer. Towards robust distributed systems (abstract). In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '00, pages 7–, New York, NY, USA, 2000. ACM.
- [7] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, 1(2):1277–1288, Aug. 2008.
- [8] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's globally-distributed database. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 251–264, Berkeley, CA, USA, 2012. USENIX Association.
- [9] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 205–220, New York, NY, USA, 2007. ACM.
- [10] J. Du, S. Elnikety, A. Roy, and W. Zwaenepoel. Orbe: Scalable causal consistency using dependency matrices and physical clocks. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 11:1–11:14, New York, NY, USA, 2013. ACM.
- [11] T. Hoff. Latency is everywhere and it costs you sales - how to crush it. <http://highscalability.com/latency-everywhere-and-it-costs-you-sales-how-crush-it>, 2009.
- [12] T. Kraska, G. Pang, M. J. Franklin, S. Madden, and A. Fekete. Mdcc: Multi-data center consistency. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 113–126, New York, NY, USA, 2013. ACM.
- [13] A. Lakshman and P. Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, Apr. 2010.
- [14] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.
- [15] C. Li, J. Leitão, A. Clement, N. Preguiça, R. Rodrigues, and V. Vafeiadis. Automating the choice of consistency levels in replicated systems. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 281–292, Philadelphia, PA, June 2014. USENIX Association.
- [16] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguiça, and R. Rodrigues. Making geo-replicated systems fast as possible, consistent when necessary. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 265–278, Berkeley, CA, USA, 2012. USENIX Association.
- [17] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Don't settle for eventual: Scalable causal consistency for wide-area storage with cops. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 401–416, New York, NY, USA, 2011. ACM.
- [18] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Stronger semantics for low-latency geo-replicated storage. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pages 313–328, Berkeley, CA, USA, 2013. USENIX Association.
- [19] P. Mahajan, S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish. Depot: Cloud storage with minimal trust. *ACM Trans. Comput. Syst.*, 29(4):12:1–12:38, Dec. 2011.
- [20] P. E. O'Neil. The escrow transactional method. *ACM Trans. Database Syst.*, 11(4):405–430, Dec. 1986.
- [21] N. Preguiça, J. L. Martins, M. Cunha, and H. Domingos. Reservations for conflict avoidance in a mobile database system. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, MobiSys '03, pages 43–56, New York, NY, USA, 2003. ACM.
- [22] E. Schurman and J. Brutlag. Performance related changes and their user impact. Presented at velocity web performance and operations conference, 2009.
- [23] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. Conflict-free replicated data types. In *Proceedings of the 13th International Conference on Stabilization, Safety, and Security of Distributed Systems*, SSS'11, pages 386–400, Berlin, Heidelberg, 2011. Springer-Verlag.
- [24] L. Shriram, H. Tian, and D. Terry. Exo-leasing: Escrow synchronization for mobile clients of commodity storage servers. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '08, pages 42–61, New York, NY, USA, 2008. Springer-Verlag New York, Inc.
- [25] S. Sivasubramanian. Amazon dynamodb: A seamlessly scalable non-relational database service. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 729–730, New York, NY, USA, 2012. ACM.
- [26] Y. Sovran, R. Power, M. K. Aguilera, and J. Li. Transactional storage for geo-replicated systems. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 385–400, New York, NY, USA, 2011. ACM.