Towards Fast Invariant Preservation in Geo-replicated Systems

Valter Balegas, Sérgio Duarte
Carla Ferreira, Rodrigo Rodrigues
Nuno Preguiça
NOVA-LINCS
DI/FCT/Univ. Nova de Lisboa

Mahsa Najafzadeh, Marc Shapiro
Inria & UPMC-LIP6

ABSTRACT

Today's global services and applications are expected to be highly available, scale to unprecedented number of clients and offer reliable, low-latency operations. This can be achieved through geo-replication, especially when data consistency is relaxed. There are however applications whose data must obey global invariants at all times. Strong consistency protocols easily address this issue but require global coordination among replicas and inevitably degrade application throughput and latency.

While coordination is an inherent requirement for maintaining global application invariants, there are instances where coordination on a per operation basis can be avoided. In particular, it has been shown that either moving coordination outside the critical path for executing operations, or having one coordination round for multiple operations, are both effective ways to maintain global invariants and avoid most of the penalties of coordination. Our stance is that expanding this idea to geo-replicated settings has yet to be fully realised.

In this paper, we review the design space of current solutions for engineering geo-replicated applications and present our guiding vision towards a general technique for providing global application invariants under eventual consistency, as a much cheaper alternative to strong consistency.

1. INTRODUCTION

The advent of global Internet-based services and applications has fuelled the rise of cloud computing and exposed the challenges of building distributed applications targeting millions of users scattered across the globe. Turning users into customers or potential customers of a whole new economy of social networks and e-commerce platforms made quality of service paramount to achieve success online.

A measure of quality of service that users perceive directly is the responsiveness of their interactions with the service. There is evidence from major industry players [29, 15, 25] that even a slight degradation in latency correlates with increased user dissatisfaction and, consequently, loss of revenue. In recent years, a great deal of research and technology advances have been directed to addressing this issue.

Geo-replication is a widely adopted technique to improve the responsiveness of online services. It employs multiple data centers, placed at strategic locations across the globe, and attempts to redirect user requests to a nearby replica of the service. Thus, the latency between end-users and the servers can be significantly reduced, in addition to offering improvements in system scalability and fault tolerance.

Under geo-replication, systems scale-out by partitioning data requests [14, 5, 12, 18]. Yet, the need to replicate databases over high latency, intercontinental network links forces system designers to choose between system availability and data consistency, since it is not possible to have both under network partitions [10]. Eventually consistent and strongly consistent systems are at the opposite extremes of that trade-off.

Eventually consistent systems forgo tight replica coordination to favor availability, allowing replicas to diverge under network partitions. Operations are executed locally and their effects are replicated asynchronously. This allows users to observe the immediate effects of their actions, but can result in concurrency anomalies, due to conflicting operations performed at other sites. In order to maintain global invariants, applications on top of eventually consistent data stores require additional programming logic, thus complicating their design and development.

Strongly consistent systems, in contrast, are well suited for applications that need to enforce global application invariants across replicas, at all times [9]. In these systems, data consistency is achieved by limiting concurrency, either by funnelling all updates to a central site, or running some consensus algorithm, such as Paxos, so that all sites agree on some global order of operations. However, performing this level of coordination every time the application state is mutated is expensive, particularly in the case of replicas that are far apart, as expected in geo-replication settings. In either case, throughput and scalability are compromised.

In an attempt to bridge the gap between availability and consistency, researchers sought to figure out what guarantees are attainable without impairing availability [3, 34]. They determined that, under some conditions, Causality is the strongest form of always-available consistency [3]. It also happens this is insufficient for enforcing global application invariants, such as ensuring non-negativity of an inventory counter under concurrent decrements.

Others have pursued the approach of combining the best aspects of eventual and strong consistency into systems that choose the most appropriate consistency level for each of the workload operations [33, 22]. Whether that choice is made manually by the programmer (a delicate and error prone process) or by a tool [21, 11, 19], it still remains that the strongly consistent execution path can still undermine availability and performance if those operations are frequent.

While coordination is necessary for enforcing global invariants under concurrency, it should be possible to reap additional parallelism from the following observation: in many cases, operations that in general are unsafe under concurrency, only actually break invariants when particular limit conditions are reached. For instance, when a non-negative counter is far from zero, concurrent decrements do not produce anomalous behaviour, regardless of the order they are committed to the database. In other cases, the typical frequency of unsafe operations in a given application workload may provide an opportunity to save on coordination costs. For instance, the frequency of operations that imperil a referential integrity invariant may be tiny compared to the rest of operations. Treating all these operations in the same way may miss the chance for optimizations - for instance, by requiring the rarer operation to perform most of the burden of global coordination may allow executing the most frequent operation without need to contact other replicas in the common case. These insights have motivated us to improve geo-replication performance in a principled way by moving coordination outside the critical execution path of operations, instead of focusing on the ordering of operations – the approach that is employed by most existing solutions.

The rest of the paper is structured as follows. In section 2, we further discuss the limitations of eventual consistency (EC) using a social network application as an example. Section 3 covers some work on using program analysis to determine which operations require coordination to ensure invariants; then, in Section 4 we review additional techniques for enforcing invariants. Section 5 presents the overall approach of our work for providing global application invariants on top of eventual consistency. Section 6 concludes the paper.

2. PITFALLS OF EVENTUAL CONSISTENCY

Eventual consistency guarantees that *in the future, if updates cease, all replicas will converge to the same value, becoming indistinguishable* [35]. In systems that offer eventual consistency, clients can access any replica, which allows the system to provide high availability despite failures as long as a single replica is available. Additionally, these systems tend to achieve low latency, as the client can access the closest replica. These advantages come at the price of increased complexity in application design [32].

In this section, we use a social network application to illustrate the anomalies that can occur in eventually consistent systems and how to address them by requiring additional guarantees from the system.

In particular, we start by discussing session guarantees [34], which are an interesting set of additional guarantees that can be implemented by eventually consistent systems.

In a social network, a user writes posts that are added to her own wall and to the walls of all her friends. We say that the system provides the *monotonic reads* session guarantee [34] if, after observing some post, successive read operations return a state that includes the post (unless it was explicitly removed). The system provides *read-your-writes* if the client will always reads her previous posts. These guarantees respect only a single user session and can be supported by requiring stick-availability, in which a client maintains *stickness* or *affinity* with a server (or set of servers) [3] or acts as a server by caching the writes and returning them in subsequent reads.

Other session guarantees concern the state observable by any client session. In particular, the system provides the *monotonic writes* guarantee if when a client executes two successive writes, any read that includes the effects of last write also include the effects of the first write.

The final session guarantee is motivated by the fact that, when a post is a reply to a previous post, a user expects to observe the original post before the reply. A system providing the *writes follows reads* guarantee enforces this property – more precisely, if a client does a write w after observing the effects of a set of previous writes S_w, any client that observes the effects of w will also observe the effects of S_w. A system that enforces causality [20] guarantees that all these sessions guarantees are respected, as events are delivered to different replicas according to the happens-before relation. Many recent systems provide causal consistency [23, 1, 24, 38, 5].

In addition to session guarantees, there are other interesting properties that eventually consistent systems may decide to provide. For example, consider the following set of requirements. In social network systems, friendship is usually a bidirectional relation, i.e., if user A is a friend of user B, user B is also friend of user A. As such, when a friend request is acknowledged, both friend lists must be updated. Updating the friend lists without atomicity may result in some user observing that A is friend of B but B is not friend of A or vice-versa. This violates the friendship relation invariant. To address this, some geo-replicated systems provide atomicity for a sequence of writes, while enforcing causality [24, 38, 33].

Finally, as a more challenging requirement, consider the following example scenario. Social networks allow the creation of groups where users can interact. The only invariant that tends to exist is that a user can only join a group for which she has been invited. This rule is easily enforced by using causal consistency, which guarantees that the acceptance of an invitation will always follow the invitation itself. However, stricter semantics would be impossible to enforce relying only on causal consistency, particularly when concurrent operations can lead to a state where the invariant is violated. For example, it is impossible to guarantee that every member of the group is friend of the administrator of the group, since a friendship relationship could be cancelled while a user concurrently joins the group. This invariant can instead be repaired after the violation is detected – e.g., by removing from the group the members that are no longer friends of the administrator.

However, some other invariants may not have a trivial repair function – consider that an award is given to a limited number of users in the group. A system relying on causal consistency could concurrently give out more awards than the limit. In this case, there is no trivial solution to select the users that should remain in the set of awardees, and the situation can be particularly problematic in case the award emails have been sent out.

The examples presented in this section show that there are several additional guarantees that eventually consistent systems should provide. However, in some cases these guarantees can be particularly difficult to enforce under eventual consistency, even with the help of a repair function. As such, to address these requirements, applications tend to adopt strong consistency models (or at least provide support for both weak and strong semantics [33, 22]).

3. MAKING THE RIGHT CHOICE

In the previous section we have seen that not all operations have the same consistency requirements. For this reason, many existing systems take the approach of supporting different levels of consistency to implement applications efficiently.

Gemini [22], Bloom^L [11], Walter [33] and Lazy Replication [19] allow developers to choose between different levels of consistency to ensure application correctness. This approach allows developers to use eventual consistency when operations are compatible with any possible concurrent updates, and only use strong consistency when concurrent operations can make the database inconsistent. This allows for fine tuning the consistency requirements of each operation. However, it poses an heavy burden on the programmer, who must decide the correct level of consistency to use: if the programmer is too conservative, this may lead to an inefficient application; if the programmer is too relaxed due to incorrect reasoning about the application semantics, this can lead to incorrect behaviour. Recent work has proposed to identify the best consistency level automatically, which provides good results free of human error. In particular, Sieve [21] determines the consistency level for operations that run on top of Gemini, under RedBlue consistency. It combines static and dynamic analysis to determine which operations are safe under causal consistency, and which operations need serializability to maintain invariants. The analysis considers a set of user-provided invariants and small annotations that specify the convergence techniques used for concurrent operations on the same objects.

The first step of the analysis, completed offline, generates abstract models that represent the space of possible concurrent executions during runtime and, for each model, determines the set of minimal pre-conditions for being safe to execute the operation without coordination.

At runtime, an operation executes under causal consistency if the minimal pre-conditions for weak execution determined offline are matched. Otherwise, the operation executes under strong consistency.

For example, the offline algorithm would determine that any operation that adds a negative value to a non-negative stock is unsafe to be executed under eventual consistency (as concurrent operations can lead the stock to become negative). At runtime, if an operation adds a positive value, it will execute under eventual consistency; otherwise it needs to execute under strong consistency.

Bloom^L[11] is a logic programming language for distributed applications that maintains application invariants. It is based on the observation that monotonic programs never retract information that is previously known, and therefore they converge regardless the delivery order of messages in different replicas. A total order of messages is only required for non-monotonic operations. An important part of Bloom^L is the CALM analysis that allows to identify which parts of the program are non-monotonic.

The Bloom^L language provides a library of semi-lattice constructs that ensure convergence, similar to CRDTs[30]. The language supports non-monotonic operators: operators that may give different results depending on the arrival order of remote messages. For executing a non-monotonic operator, a coordination protocol must be executed, to ensure that the result of the non-monotonic operation is equivalent in all replicas.

Both strategies identify which operations may break invariants and require coordination among replicas to execute them. This strategy is conservative, as in many executions it is safe to execute the operations without coordination. For instance, in the stock example, coordination is only necessary when the number of available units becomes low, but the system is forced to coordinate on every request because it does not take the current level of the stock into account.

When determining if an operation can execute without coordination. Bloom^L looks only at the code of operations, while Sieve takes into consideration both the code of the operation and the value of parameters. In the latter case, the final decision on whether coordination is necessary or not is executed in runtime. We argue that it is possible to extend this approach by considering also the state of the database. This has the potential to reduce, or even completely avoid, the cost of global coordination by extending conflict analysis with runtime information about the database and the participants.

In the literature, some proposals use the estimation of replica divergence to avoid coordination [37, 17], either by using deterministic or stochastic models. However, these techniques cannot be applied to general invariants and only give a estimation of the divergence, allowing invariants to be broken in certain scenarios.

4. OLD TECHNIQUES REVISITED

In this section, we revisit two works that inspire our vision for enforcing invariants without requiring coordination in the critical path of operation execution: the escrow transactional method [26] and the demarcation protocol [7]. We discuss the use of these protocol to provide the invariants from Section 2 without using strong consistency, or replica coordination in the general case.

The escrow model [26] was proposed to allow long-lived transactions to commit without interfering with other ongoing transactions. The key idea is to divide resources into *escrows* that can be used concurrently by different nodes. If the client has enough resources in its escrow, it can execute the operation without coordination and release the remaining resources on commit, or abort.

In the example of the limited number of awards, consider that each group has a limit of K awards. Each node i that holds a copy of group G grants awards up to a limit Y_i such that $\sum_{i=1}^{n} Y_i <= K$, where n is the number of copies of G. While the number of given awards do not exceed the local limit Y_i, each node can execute the operations locally with low-latency.

This model has been extended to support different partitionable data types [36] and operations [27, 31], but all implementations rely on a central component to manage escrows.

The demarcation protocol [7] has a insight similar to the escrow model, but enforces invariants over multiple variables. For each variable, the protocol defines a limit for the value of the variable. The combination of the defined limits for all variable guarantee that the defined invariants remain valid. Thus, operations are safe if updates do not exceed the defined limits.

If an operations requires a variable to exceeds its limit, another peer must change its limit to make that operation safe: a node sends a request with the change in the limit it requires; the node that accepts the request adjust its own limits and notifies the requester of the change; the requester then increase its safety limits with the received delta and the operation executes safely.

Changing the limits with point-to-point communication can be fast when nodes know enough information about the other peers. When the resources are scarce and nodes change the limits more frequently some request might fail leading to multiple point-to-point messages. Additionally, the point-to-point protocol needs to enforce exactly-once delivery or the limits may become more restrictive than necessary.

The authors have used this protocol to maintain a numeric invariant over resources distributed in multiple machines, enforcing the uniqueness invariant and to provide referential integrity constraints.

A referential integrity constraint is modelled by a logical implication: $predicate(A) \Rightarrow predicate(B)$. Each nodes stores a boolean value for each predicate. The idea is to enforce that whenever a node updates a predicate to a value that may turn the expression false (unsafe), it must enforce that the other nodes changes the value of their predicate to maintain the expression true. In our example, we have $JoinGroup(A, G) \Rightarrow isFriend(A, B)$, with A a user, B the administrator and G a group of users[1]. Making $JoinGroup(A, G)$ *true* is unsafe because that value is only allowed if $isFriend(A, B)$ is true, otherwise the expression is false. The node requests the peer holding the predicate $isFriend(A, B)$ to change the minimum value for that predicate to true. The converse must also be ensured, to make $isFriend(A, B)$ *false* - the node must ask the peer holding the value for predicate $JoinGroup(A, G)$ to ensure it is false.

The idea of distributing data by multiple nodes in an infrastructure has been widely adopted in other contexts to do load balancing for distributed memory multiprocessor [13], quota enforcement in grid [16] and cloud environments [8]. More recently MDCC [18] uses a variation of the demarcation protocol to extract more concurrency of commutative operations that maintain numerical constraints invariants. The homeostasis protocol [28] also extends the demarcation protocol, but requires a new set of conditions to be computed and installed in all replicas using two-phase commit. We argue that it is possible to leverage these old ideas in the new geo-replicated settings relying on peer-to-peer and unreliable asynchronous communication protocols only, as discussed in the next section.

5. LOW-COST INVARIANTS

In the previous sections, we have shown techniques that allow the maintenance of database invariants in two different ways: by identifying what operations are not safe and use strong consistency to execute those or by enforcing local constraints to ensure that operation are safe, while the system is divergent. We argue that a combination of these techniques can be used to provide a principled approach to execute operations that maintain invariants without coordination in the general case.

We envision a system that identifies operations that require strong consistency, but use an efficient protocol to guarantee that local executions are safe instead of using global coordination. The system would exchange the necessary resources, outside the critical path of execution, to guarantee that operations can execute safely, but could still resort to strong consistency when those requirements are not met.

Our preliminary investigations indicates that Sieve is a good candidate to build our system. We could modify the analysis that determines the weakest pre-conditions to accept more facts, computed during runtime, to enable the execution of more operations locally. For instance if the weakest precondition to execute $joingroup(A, G)$ is that $isFriend(A, B)$, than we could add some fact that gives the local replica the exclusive right to modify that predicate, which would ensure it does not become false. On execution, if the current replica holds that guarantee it can execute the operation without coordination because it has the guarantee that the value of that predicate can only change locally. Otherwise, it should resort to strong consistency to execute the operation.

[1]The invariant presented is simplified for illustration purpose, it should also ensure that B is administrator of G

We have a preliminary design of a data-type that maintains numerical invariants [6]. Our data-type maintains the full state of the invariant, which allows the current value to be queried by a client, in opposition to the demarcation protocol, that may require to contact multiple nodes before knowing the actual value of the inequality. Evaluation shows that all operations execute locally while they do not contend for the last available resources. This is a first stepping stone to provide data-types that are able to preserve the demarcation protocol invariants in a replicated system.

Our approach is able to maintain different forms of invariants and we already have a data-type that realises the numerical invariants, but it remains an open question what is the extent of invariants can we capture. Baillis et Al. [4] made a survey on the typical invariants on benchmarks and concluded that the most common invariants have the form of referential integrity, numerical constraints and uniqueness, which all can be implemented with the demarcation protocol.

To our knowledge, none of the previous approaches can be directly applied to implement our vision. None of the former works addresses all the key points in building geo-replicated data-bases: Either they only capture limited forms of invariants, do not deal with data-replication, rely on a central components to manage resources or do not provide low-latency, fault tolerance and scalability to million of clients.

6. CONCLUSION

Current systems give up low-latency and availability for consistency when invariants are essential to applications. At best, only those invariants that are compatible with eventual consistency can be enforced with low latency. For the rest, the default has been to rely blindly on strong consistency. To help figure out which case applies, recent research has produced techniques that help programmers sort out which parts of a program are unsafe under concurrency and need global coordination. Avoiding coordination over expensive inter-continental links has proved to be an important optimisation with noticeable impact on performance.

In this paper, we propose leveraging additional techniques to further avoid paying the full cost of coordination while enforcing global invariants on top of eventual consistency. To the best of our knowledge, no current implementations are tailored to harness these techniques on cloud infrastructures.

After reviewing the literature, we concluded that the approach applies to the most frequent application invariants. We have already confirmed this in part with the design of a data-type that maintains numerical invariants with low-latency. We are now adapting these protocols to be deployed on geo-replicated systems and have been using existing analysis techniques to determine when our optimizations can be applied.

Acknowledgments

This research is supported in part by European FP7 project 609 551 SyncFree (2013–2016), Fundação para a Ciência e Tecnologia SFRH/BD/87540/2012 and PEst-OE/ EEI/ UI0527/ 2014.

7. REFERENCES

[1] S. Almeida, J. 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. Davidson, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Highly available transactions: Virtues and limitations. *PVLDB*, 7(3):181–192, 2013.

[3] P. Bailis, A. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Coordination-avoiding database systems. *CoRR*, abs/1402.2237, 2014.

[4] 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.

[5] V. Balegas, M. Najafzadeh, S. Duarte, C. Ferreira, rodrod, M. Shapiro, and N. Preguiça. The case for fast and invariant-preserving geo-replication. In *W-PSDS 2014: Workshop on Planetary-Scale Distributed Systems, 2014*. IEEE Computer Society, 10 2014.

[6] 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.

[7] J. Behl, T. Distler, R. Kapitza, and T. Braunschweig. Dqmp: A decentralized protocol to enforce global quotas in cloud environments. In *In Proc. of SSS '12*, pages 217–231, 2012.

[8] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[9] 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.

[10] N. Conway, W. R. Marczak, P. Alvaro, J. M. Hellerstein, and D. Maier. Logic and lattices for distributed programming. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 1:1–1:14, New York, NY, USA, 2012. ACM.

[11] 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.

[12] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *J. Parallel Distrib. Comput.*, 7(2):279–301, Oct. 1989.

[13] 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.

[14] P. Dixon. Shopzilla's site redo - you get what you measure. Presented at velocity web performance and operations conference, 2009.

[15] K. Karmon, L. Liss, and A. Schuster. Gwiq-p: An efficient decentralized grid-wide quota enforcement protocol. *SIGOPS Oper. Syst. Rev.*, 42(1):111–118, Jan. 2008.

[16] T. Kraska, M. Hentschel, G. Alonso, and D. Kossmann. Consistency rationing in the cloud: Pay only when it matters. *Proc. VLDB Endow.*, 2(1):253–264, Aug. 2009.

[17] 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.

[18] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat. Providing high availability using lazy replication. *ACM Trans. Comput. Syst.*, 10(4):360–391, Nov. 1992.

[19] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

[20] C. Li, J. a. Leitão, A. Clement, N. Preguiça, R. Rodrigues, and V. Vafeiadis. Automating the choice of consistency levels in replicated systems. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, pages 281–292, Berkeley, CA, USA, 2014. USENIX Association.

[21] 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.

[22] 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.

[23] 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.

[24] M. Mayer. In search of... a better, faster, stronger web. Presented at velocity web performance and operations conference, 2009.

[25] P. E. O'Neil. The escrow transactional method. *ACM Trans. Database Syst.*, 11(4):405–430, Dec. 1986.

[26] 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.

[27] S. Roy, L. Kot, N. Foster, J. Gehrke, H. Hojjat, and C. Koch. Writes that fall in the forest and make no sound: Semantics-based adaptive data consistency. *CoRR*, abs/1403.2307, 2014.

[28] E. Schurman and J. Brutlag. Performance related changes and their user impact. Presented at velocity web performance and operations conference, 2009.

[29] 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.

[30] L. Shrira, 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.

[31] J. Shute, R. Vingralek, B. Samwel, B. Handy, C. Whipkey, E. Rollins, M. Oancea, K. Littlefield, D. Menestrina, S. Ellner, J. Cieslewicz, I. Rae, T. Stancescu, and H. Apte. F1: A distributed SQL database that scales. *PVLDB*, 6(11):1068–1079, 2013.

[32] 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.

[33] D. B. Terry, A. J. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. W. Welch. Session guarantees for weakly consistent replicated data. In *Proceedings of the Third International Conference on Parallel and Distributed Information Systems*, PDIS '94, pages 140–149, Washington, DC, USA, 1994. IEEE Computer Society.

[34] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, SOSP '95, pages 172–182, New York, NY, USA, 1995. ACM.

[35] G. D. Walborn and P. K. Chrysanthis. Supporting semantics-based transaction processing in mobile database applications. In *Proceedings of the 14TH Symposium on Reliable Distributed Systems*, SRDS '95, pages 31–, Washington, DC, USA, 1995. IEEE Computer Society.

[36] H. Yu and A. Vahdat. Design and evaluation of a continuous consistency model for replicated services. In *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation - Volume 4*, OSDI'00, pages 21–21, Berkeley, CA, USA, 2000. USENIX Association.

[37] M. Zawirski, A. Bieniusa, V. Balegas, S. Duarte, C. Baquero, M. Shapiro, and N. M. Preguiça. Swiftcloud: Fault-tolerant geo-replication integrated all the way to the client machine. *CoRR*, abs/1310.3107, 2013.