Making Weak Consistency Great Again

Valter Balegas, Sérgio Duarte
Carla Ferreira, Nuno Preguiça
NOVA LINCS & DI, FCT, Universidade NOVA de Lisboa

Rodrigo Rodrigues
INESC-ID & IST, University of Lisbon

ABSTRACT

This paper focuses on the problem of implementing web applications on top of weakly consistent geo-replicated systems. Several techniques, such as CRDTs, have been proposed to achieve state convergence on a per-object and per-data type basis. However, that does not guarantee application correctness, as convergence rules applied individually at each object may lead to an invalid state.

We advocate that it is possible to address these problems and implement correct applications under weak consistency. To that end, it is necessary to combine CRDTs with novel semantics, judiciously select the CRDTs that are used by applications, and transform application operations to guarantee that convergence rules, applied on a per-object basis, always lead to valid application states. Achieving this is complex and requires tools to help programmers tame the complexity of programming on top of weak consistency and make the technology more accessible.

In the presentation of this work we make a demonstration of a prototype tool that is capable of detecting concurrency conflicts on applications and propose transformations to make them conflict-free.

Keywords

Geo-Replication, Weak Consistency, CRDT

1. INTRODUCTION

The pervasiveness of the Internet in people's day-to-day activities led to a paradigm shift in the way developers build web applications. Nowadays, systems need to scale to unprecedented levels and provide good quality of service to users across the globe. Centralized systems are inherently hard to scale, prone to failure, and provide poor quality of service for users that access it from remote locations. This led developers to become more interested in developing distributed systems that do not suffer from such limitations. In this context, geo-replication appears as a core technique to implement scalable and responsive distributed systems for clients scattered across the globe.

Systems that use geo-replication typically provide weaker forms of consistency in order to allow replicas to process requests without contacting remote replicas. As a consequence, when the same objects are updated at different locations, their values diverge and need to be reconciled later. Conflict-free Replicated Data types (CRDTs) [10] are a principled approach to handle replica convergence. These data types support the reconciliation of multiple divergent copies of the same object, with a well-defined semantics. Various systems use CRDTs to provide richer programming models for developers [5, 11].

Despite the effort to make weak consistency systems easier to program, this is still a challenging task. In systems that use strong consistency, application correctness is always ensured as long as the application's individual operations respect the invariants. However, this is not the case for applications implemented on top of weak consistency, where developers must carefully ensure that application invariants hold under concurrent execution. The root of this problem is that the reconciliation of concurrent updates might produce a state that is invalid with respect to the application invariants. Bailis et al. studied several applications available online and found that many applications built on top of weak consistency do not provide the expected semantics [2].

In our recent work with bounded counter [4], we designed a counter that can maintain numeric invariants under the execution of concurrent updates. The same strategy can be used with other data types to provide other invariants. However a more general problem remains unsolved: how to maintain invariants across multiple objects, without loss of availability? An example of such invariants is to ensure referential integrity in relational databases. Previous research addressed this issue by constraining the operations that can be executed in each replica in order to preserve data integrity [3, 8]. Whenever a replica receives a request to execute an operation that might violate an invariant, the replica must coordinate with remote replicas to ensure that the operation is safe. Although this coordination may sometimes be executed outside of the critical execution path of operations, a replica can always be prevented from executing some operation because it needs to contact a remote replica that is unavailable.

In this paper, we study the example of referential integrity and show how to transform an application to provide that invariant on top of weak consistency. We discuss different semantics to solve the conflict without constraining concur-

PaPOC'16, April 18-21 2016, London, United Kingdom

ISBN 978-1-4503-4296-4/16/04...$15.00

DOI: http://dx.doi.org/10.1145/2911151.2911167

rency, showing that it is possible to implement scalable and correct applications on top of weak consistency. We also present the current status of a tool we are developing to help developers in that process.

2. RUNNING EXAMPLE

We chose referential integrity as running example due to its importance in relational databases and concurrent programming in general. We consider a toy database composed of two entities, A and B. We assume, without loss of generality, that each entity has a single attribute. There is a one-to-many relationship $R_{a\to b}$ from elements of A to elements of B.

Consider the implementation of this example using an object-relational mapping approach, where entities are modeled as two distinct sets and the relationship between them are modeled by a third set of pairs $(a, b) : a \in A, b \in B$.

We assume that the storage system stores each set in separate objects and that it provides causal consistency and atomic updates across multiple objects.

The integrity constraint of this model is broken when $\exists(a, b) \in R_{a\to b} : a \notin A \vee b \notin B$, i.e., there is a relationship between entity a and b, but one or both of them do not exist. We consider, for simplicity, that the application is correct under strong consistency, i.e., any sequential execution of the program does not violate the invariant. An invariant violation only occurs when a client issues an operation to create a new relation (a, b) while another client issues an operation to remove a or b from A or B, respectively.

3. BETTER SAFE THAN SORRY

To allow fast execution without constraining concurrency, every replica must be able to reply to a request without depending on remote state. Under these circumstances it is not possible to avoid concurrent executions that might leave the database in an inconsistent state. Since detecting conflicts and fixing an invalid database state is expensive, we propose solving conflicts beforehand, so that execution is always safe. The idea is that an operation can have extra effects in order to avoid generating an invalid state when replicas are reconciled. As a trade-off, the semantics of operations that are implemented this way is limited, but, as we show next, interesting semantics can be provided with proper use of convergence rules.

In the next section we describe two alternative solutions for the problem we described above. In the first solution we rely exclusively on existing CRDT semantics, while in the second solution we devise a new convergence rule for concurrent operations to implement an alternative semantics.

3.1 Adding missing elements

When a new element (a, b) is added to $R_{a\to b}$, the operation that adds this element to the set of relations must ensure that $a \in A$ and $b \in B$ to preserve referential integrity. These elements might be removed concurrently at other replicas leading to an invariant violation after replicas reconcile. To avoid this conflict, we modify the operation that adds (a, b) to $R_{a\to b}$ to also add a to A and b to B, atomically, and set the convergence rule of each set to use an Add-Wins policy [10]. This policy ensures that if an add and remove operations execute concurrently for the same element, then the element will be present in the set, cancelling the effects of the concurrent remove operation. The consequence of these modifications is that any operation to remove elements a or b will be cancelled by the additional effect of the operation that adds (a, b), if they execute concurrently.

3.2 Ensuring that elements are removed

In the previous solution, whenever the conflicting operations execute, the operation that adds the element to $R_{a\to b}$ takes precedence over the remove operations. We might also want the opposite semantics, i.e., that whenever a remove operation for a or b is issued, we cancel any concurrent operation that adds an element to $R_{a\to b}$ containing one of those values. This example is different from the previous one and cannot be solved in the same way, because we do not know the possible pairs containing a or b that might be added to the set, and it would be too expensive to consider the whole domain of A or B. In the presentation, we present a new set CRDT that prevents concurrently adding elements to a set that match a given criteria, without specifying their values.

The intuition behind this new set is to provide a special *touch(Predicate p)* operation that accepts a predicate that specifies which elements we want to prevent adding concurrently to the set. This way, whenever we execute a remove operation for elements a or b, we also execute a *touch* in $R_{a\to b}$ that prevent the addition of any pair matching $(a, *)$ or $(*, b)$, where $*$ means any element of B.

4. TOOLS FOR PROGRAMMING WEAK CONSISTENCY

In the previous section we saw how to preserve referential integrity in applications developed on top of weak consistency. Even though the transformations to the operations are easy to explain, it might be challenging for programmers to devise them. For this reason, we are also working on tools that can ease the identification of invariant violations in applications and propose possible solutions.

We are building a tool that, given the specification of an application's operations and invariants, identifies conflicts that might arise due to concurrent executions and proposes transformations to the operations to fix them, without strengthening the consistency model that is employed. The algorithm for identifying conflicts has already been published in our previous work [3]. We are extending this tool to propose transformations to the operations like the ones we described before. In the presentation we show the tool in action to solve the referential integrity example.

5. FINAL REMARKS

In this paper we have presented an overview of how to maintain a specific global invariant over multiple objects in a weakly consistent replicated system, by making a judicious use of replicated data type semantics. Using a similar approach in an application is in general rather complex for programmers, which need to reason about concurrent executions on weakly consistent systems and on the semantics of convergent replicated data types. To address this problem, we have implemented a tool for helping programmers to create applications that maintain global invariants on top of weakly consistent systems. Our tool identifies which operations can lead to concurrency issues and proposes the use of appropriate replicated data types and, when needed, changes to the operations to guarantee that invari-

ants are maintained despite the concurrent operations being executed. The tool work iteratively, receiving feedback from the programmer of which result is preferable when conflicts may arise.

Many recent systems abandoned weak consistency towards stronger consistency models, as these systems are easier to program and understand [12, 6], but this comes with costs: higher latency, lower availability and scalabilty. To mitigate these problems, several works have tried to pinpoint where weak consistency can be used and use strong consistency for everything else [1, 8, 3, 7, 9]. Despite the performance improvements for operations that remain correct under weak consistency, it does not improve operations that require strong consistency. The best performance for an applications might only be achieved if none of its operations require coordination. Our work shows that it is possible to execute a larger fraction of operations under weak consistency while still guaranteeing global invariants. To achieve this in practice, we empower programmers with a tool that helps them building applications that execute correctly under weak consistency.

Acknowledgments

This research is supported in part by EU FP7 SyncFree project (609551), FCT/MCT SFRH/BD/87540/2012, PTDC/EEI-SCR/ 1837/ 2012 and PEst-OE/ EEI/ UI0527/ 2014. The research of Rodrigo Rodrigues is supported by the European Research Council under an ERC Starting Grant.

6. REFERENCES

[1] Alvaro, P., Conway, N., Hellerstein, J. M., and Marczak, W. R. Consistency analysis in bloom: A calm and collected approach. In *In Proceedings 5th Biennial Conference on Innovative Data Systems Research* (2011), pp. 249–260.

[2] Bailis, P., Fekete, A., Franklin, M. J., Ghodsi, A., Hellerstein, J. M., and Stoica, I. Feral concurrency control: An empirical investigation of modern application integrity. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2015), SIGMOD '15, ACM, pp. 1327–1342.

[3] Balegas, V., Duarte, S., Ferreira, C., Rodrigues, R., Preguiça, N., Najafzadeh, M., and Shapiro, M. Putting consistency back into eventual consistency. In *Proceedings of the Tenth European Conference on Computer Systems* (New York, NY, USA, 2015), EuroSys '15, ACM, pp. 6:1–6:16.

[4] Balegas, V., Serra, D., Duarte, S., Ferreira, C., Shapiro, M., Rodrigues, R., and Preguiça, N. M. Extending eventually consistent cloud databases for enforcing numeric invariants. In *34th IEEE Symposium on Reliable Distributed Systems, SRDS 2015, Montreal, QC, Canada, September 28 - October 1, 2015* (2015), pp. 31–36.

[5] Basho. Riak. http://basho.com/riak/, 2014. Accessed Fev/2016.

[6] Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J. J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., Hsieh, W., Kanthak, S., Kogan, E., Li, H., Lloyd, A., Melnik, S., Mwaura, D., Nagle, D., Quinlan, S., Rao, R., Rolig, L., Saito, Y., Szymaniak, M., Taylor, C., Wang, R., and Woodford, D. Spanner: Google's globally-distributed database. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2012), OSDI'12, USENIX Association, pp. 251–264.

[7] Gotsman, A., Yang, H., Ferreira, C., Najafzadeh, M., and Shapiro, M. 'cause i'm strong enough: Reasoning about consistency choices in distributed systems. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (New York, NY, USA, 2016), POPL 2016, ACM, pp. 371–384.

[8] Li, C., Porto, D., Clement, A., Gehrke, J., Preguiça, N., and Rodrigues, R. Making geo-replicated systems fast as possible, consistent when necessary. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2012), OSDI'12, USENIX Association, pp. 265–278.

[9] Roy, S., Kot, L., Bender, G., Ding, B., Hojjat, H., Koch, C., Foster, N., and Gehrke, J. The homeostasis protocol: Avoiding transaction coordination through program analysis. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2015), SIGMOD '15, ACM, pp. 1311–1326.

[10] Shapiro, M., Preguiça, N., Baquero, C., and Zawirski, M. Conflict-free replicated data types. In *Proceedings of the 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)* (Grenoble, France, Oct. 2011), X. Défago, F. Petit, and V. Villain, Eds., vol. 6976 of *Lecture Notes on Computer Science*, Springer, pp. 386–400.

[11] Zawirski, M., Preguiça, N., Duarte, S., Bieniusa, A., Balegas, V., and Shapiro, M. Write fast, read in the past: Causal consistency for client-side applications. In *Proceedings of the 16th Annual Middleware Conference* (New York, NY, USA, 2015), Middleware '15, ACM, pp. 75–87.

[12] Zhang, I., Sharma, N. K., Szekeres, A., Krishnamurthy, A., and Ports, D. R. K. Building consistent transactions with inconsistent replication. In *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP 2015, Monterey, CA, USA, October 4-7, 2015* (2015), pp. 263–278.