Dotted Version Vectors: Efficient Causality Tracking for Distributed Key-Value Stores

Nuno Preguiça, Carlos Baquero, Paulo Sérgio Almeida, Victor Fonte, Ricardo Tomé Gonçalves
nmp@di.fct.unl.pt, {cbm,psa,vff,tome}@di.uminho.pt

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA
Universidade do Minho
FCT Fundação para a Ciência e a Tecnologia
MINISTÉRIO DA CIÊNCIA E DO ENSINO SUPERIOR
COMPETE
PROGRAMA OPERACIONAL FACTORES DE COMPETITIVIDADE

Background

The design of Amazon's Dynamo system [6] was an important influence to a new generation of databases, such as Cassandra and Riak, focusing on partition tolerance, write availability and eventual consistency. These systems follow a design where the data store is always writable: replicas of the same data item are allowed to temporarily diverge and to be repaired later on. A simple repair approach, followed in Cassandra, is to use physical timestamps to arbitrate which concurrent updates should prevail. As a consequence some updates will be lost since a last writer wins (LWW) policy is enforced over concurrent updates. Thus, an approach avoiding lost updates must be able to maintain divergency until it can be reconciled.

Motivation

Accurate tracking of concurrent data updates can be achieved by a careful use of well established causality tracking mechanisms [2, 3, 4, 5]. In particular, for data storage systems, version vectors [3] enable the system to compare any pair of replica versions and detect if they are equivalent, concurrent or if one makes the other obsolete. However, current cloud storage systems, e.g. Dynamo and Riak, make several compromises regarding causality tracking, leading to lost updates and/or introduction of false concurrency. The reasoning behind these compromises is to achieve higher scalability by imposing a limit on the number of entries in the version vectors.

Dotted Version Vectors

Dotted Version Vectors is a new solution that combines a new execution model for operations with a new, and simple, causality tracking solution that allows both scalable and fully accurate tracking. The key idea of our approach is to maintain the identifier of the most recent event separate from its causal past. Besides allowing the size of information to be bounded by the degree of replication (instead of the number of clients), this approach allows to verify causality in constant time (instead of O(n) for version vectors).

References

[1] Nuno Preguiça, Carlos Baquero, Paulo Sérgio Almeida, Victor Fonte and Ricardo Tomé Gonçalves. : *Dotted version vectors: Efficient causality tracking for distributed key-value stores (2012)*, `http://gsd.di.uminho.pt/members/vff/dotted-version-vectors-2012.pdf`

[2] Lamport, L.: Time, clocks and the ordering of events in a distributed system. Communications of the ACM 21(7), 558–565 (Jul 1978)

[3] Parker, D.S., Popek, G., Rudisin, G., Stoughton, A., Walker, B., Walton, E., Chow, J., Ed- wards, D., Kiser, S., Kline, C.: Detection of mutual inconsistency in distributed systems. Transactions on Software Engineering 9(3), 240–246 (1983)

[4] Raynal, M., Singhal, M.: Logical time: Capturing causality in distributed systems. IEEE Computer 30, 49–56 (Feb 1996)

[5] Schwarz,R.,Mattern,F.:Detectingcausalrelationshipsindistrib of the holy grail. Distributed Computing 3(7), 149–174 (1994)

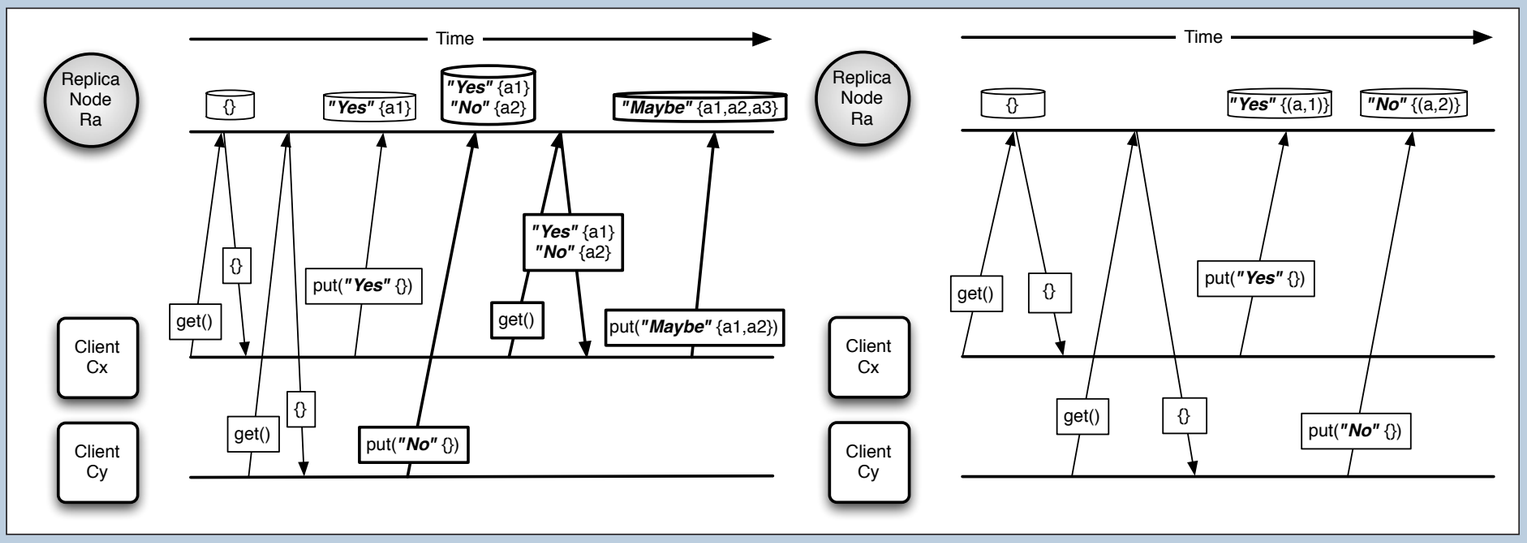
[6] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Siva- subramanian, S., Vosshall, P., Vogels, W.: Dynamo: amazon's highly available key-value store. In: pp. 205–220. SOSP '07, ACM, New York, NY, USA (2007)

Acknowledgements

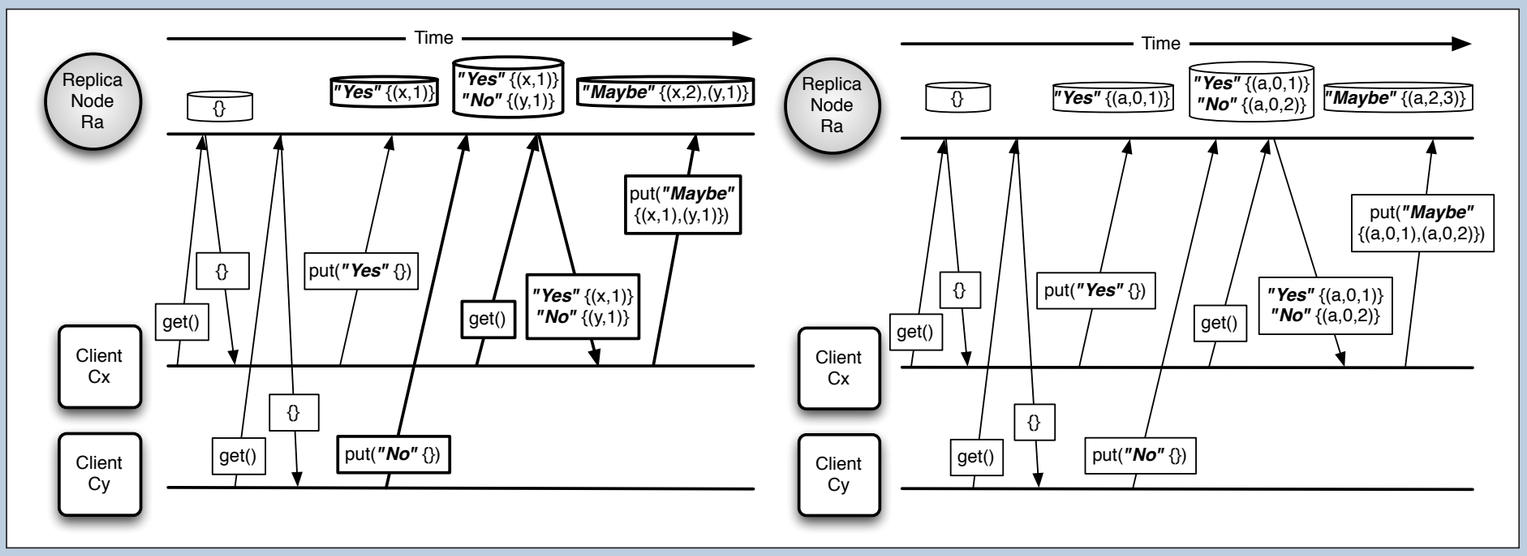
This work is funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project FCOMP–01–0124–FEDER–010114, PTDC/EIA-EIA/108963/2008 and PEst-OE/EEI/UI0527/2011.

Current Approaches

A classic version vector compresses causal histories by representing, for each component, all events up to a given sequence number. With on entry per server, this is not enough to represent the concurrent versions generated when several clients (using a standard get/put API) perform a get of some key from one server and then all perform a put, using the same causal past.
In the examples below, two clients concurrently modifying the same key on a replica node. On the left, we are using Causal Histories, and on the right Version vector with one id-per-server.



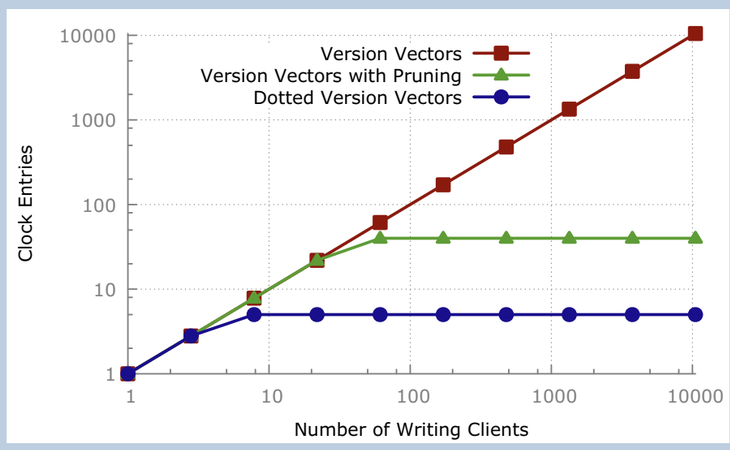
The basic idea of dotted version vectors is to add the possibility of representing an individual event (a "dot") with an isolated sequence number outside the contiguous range. This allows describing events as concurrent (due to having incomparable dots) even if they were generated from the same causal past (and have identical version vectors in the causal past).
Below we present the same example as above, but using Version vector with one id-per-client on the left and Dotted Version Vectors on the right.



Scalability and Performance

We extended and performed a set of benchmarks in Riak 0.*, and we found that the clock size is always much smaller using DVV, even with the (default) pruning that occurs with VV. We also confirm that pruning is occurring, because concurrency was higher using VV in all the tests. The difference in concurrency, results from false conflicts created by pruning. Even if the default pruning threshold was lowered in VV case, to reduce the clock size, this would also lead to an increase of false concurrency, thus a higher number of values per key.

Workload	Type	Get Mean (ms)	Get 95th (ms)	Put Mean (ms)	Put 95th (ms)	Update Mean (ms)	Update 95th (ms)	Clock Size (bytes)	Values per Key (average)
60% GET	VV	7.65	15.9	5.71	10.1	14.4	24.0	790	1.34
10% PUT	DVV	3.16	5.25	4.31	6.27	7.76	10.9	127	1.31
30% UPD	$\frac{DVV}{VV}$	**0.41**	**0.33**	**0.76**	**0.62**	**0.54**	**0.46**	**0.16**	**0.98**
30% GET	VV	10.4	21.6	7.48	13.8	18.8	31.9	859	1.20
10% PUT	DVV	3.45	5.83	4.56	6.59	8.39	11.8	123	1.16
60% UPD	$\frac{DVV}{VV}$	**0.33**	**0.27**	**0.61**	**0.48**	**0.45**	**0.37**	**0.14**	**0.97**



The figure on the left illustrates the theoretical effect of the number of writing clients in the number of clock entries, and thus the overall clock size. DVV stabilizes in size when the number of entries reaches the replication factor (here set to 3), while VV with id-per- client grow indefinitely. Thus, in practice, systems usually resort to pruning to control its growth, and lose the ability to accurately track causality.

A full description of this mechanism, and benchmarks of its integration with Riak can be found in a technical report [1].