Brief Announcement: Efficient Causality Tracking in Distributed Storage Systems With Dotted Version Vectors

Nuno Preguiça
CITI/DI-FCT-Univ. Nova de Lisboa
Portugal

Carlos Baquero, Paulo Sérgio Almeida,
Victor Fonte, Ricardo Gonçalves
HASLab, U. Minho & INESC TEC, Portugal

ABSTRACT

Version vectors (VV) are used pervasively to track dependencies between replica versions in multi-version distributed storage systems. In these systems, VV tend to have a dual functionality: identify a version and encode causal dependencies. In this paper, we show that by maintaining the identifier of the version separate from the causal past, it is possible to verify causality in constant time (instead of $O(n)$ for VV) and to precisely track causality with information with size bounded by the degree of replication, and not by the number of concurrent writers.

Categories and Subject Descriptors

C.2.4 [**Computer-communication networks**]: Distributed Systems

Keywords

Causality tracking, distributed storage systems.

1. INTRODUCTION

Tracking causality is one of the fundamental problems in distributed systems. Causality can be precisely characterized by *causal histories* [5]. Causal histories are sets of unique event identifiers. Each event, a, is assigned a new unique identifier, id_a, and its causal history, H_a, will include this identifier and the set, P_a, of identifiers for all events that causally precede a ($H_a = \{id_a\} \cup P_a$). The partial order of causality can be precisely tracked by comparing these sets by set inclusion. An history H_a causally precedes H_b iff $H_a \subset H_b$. Two histories are concurrent if neither include the other: $H_a \parallel H_b$ iff $H_a \not\subseteq H_b \land H_b \not\subseteq H_a$.

Version vectors (VV) [3] are an efficient mechanism to encode causal histories in distributed storage systems. When considering VV, unique identifiers are the composition of unique site ids and a monotonic integer counter. A version vector, V, maintains for each site, s_i, an integer $V[s_i] = n_i$ encoding that event identifiers $(s_i, 1), \ldots, (s_i, n_i)$ are included in the set represented by V (assuming that the first assigned identifier in s_i is $(s_i, 1)$). VV are used to verify the causality among replica versions: $V_a \leq V_b$, iff $\forall s, V_a[s] \leq V_b[s]$, which is no more that the application of set-inclusion defined for causal histories.

PODC'12, July 16–18, 2012, Madeira, Portugal.
ACM 978-1-4503-1450-3/12/07.

By the definition of causal history, it is clear that it is possible to verify if an event a causally precedes an event b by simply verifying if its identifier id_a is contained in the set P_b of events that precede event b : $a < b$, iff $id_a \in P_b$ (or $id_a \in H_b \land id_a \neq id_b$). Two events are concurrent if neither causally precedes the other: $a \parallel b$ iff $id_a \notin P_b \land id_b \notin P_a$. VV do not allow the use of the set-contains operation when verifying the causality dependencies of two events, as the version identifier is not known as it is diluted in the VV. In the next section, we present a causality tracking mechanism that decouples version identifiers and causality tracking information, correctly encoding the causal history.

2. DOTTED VERSION VECTORS

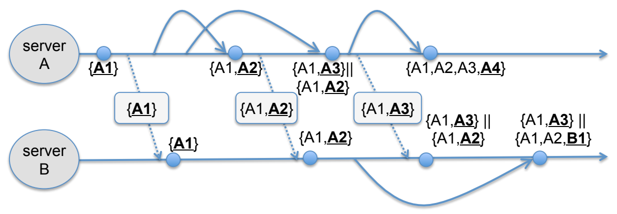
A dotted version vector (DVV) [4] is a logical clock which consists of a pair (d, v), where v is a traditional version vector and the dot d is a pair (i, n), with i a node identifier and n an integer. The dot is the version identifier and it represents the globally unique event being described, while the VV represents the causal past. Events represented by a DVV can be characterized by the following function from DVV to causal histories: $\mathcal{C}[\![((i,n), v)]\!] = \{i_n\} \cup \bigcup_j \{j_m \mid 1 \leq m \leq v[j]\}$ where i_n denotes the event with identifier (i, n).

From the definition of causal histories, it follows immediately that an event a with DVV $((i_a, n_a), v_a)$ causally precedes an event b with DVV $((i_b, n_b), v_b)$: $a < b$, iff $n_a \leq v_b[i_a]$ (i.e., the event identifier of a is in the causal past of b). Two events are concurrent if neither causally precedes the other: $a \parallel b$ iff $n_a > v_b[i_a] \land n_b > v_a[i_b]$.

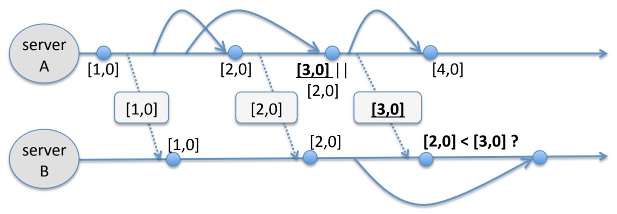
As an example, we present the evolution of the versions of an object maintained in two storage servers using both causal histories (Figure 1a) and DVV (Figure 1c). DVV are the immediate representation of causal histories, with the version identifier decoupled from the causal past. Next, we discuss the advantages and disadvantages of DVV.

$O(1)$ **causality verification:** Verifying if one event a precedes some other event b can be done in constant time with adequate data structures, by simply verifying if the event identifier (dot) of a is reflected in the causal past of b. Although in many cases this is just a theoretical curiosity, with the growing number of sites involved in distributed systems, this is becoming increasingly important.

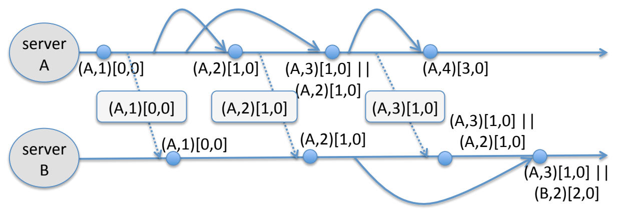
Efficient causality tracking in replicated storage systems: Distributed file systems (e.g. Locus, Coda, Ficus) usually use VV with one entry per server. This is sufficient for detecting concurrency between versions stored in servers. For detecting concurrency between the version in a server and the version a client wants to write, the client can record



(a) Causal histories (version identifier in underlined bold)



(b) Version vectors (problematic cases in underlined bold)



(c) Dotted version vectors

Figure 1: System with two servers and a single object. Client interactions are presented as curves. Server synchronizations depicted by dotted lines. The causality information maintained after each relevant event is shown close to each small circle, || meaning that concurrent versions are maintained.

the VV of the version it has read. When writing back its changes, if the VV in the server is different, a concurrent update is detected. In this case, systems as Coda require the conflict to be solved before the file can be accessed again. With DVV, conflicts can be detected by comparing only the dot, instead of the full VV – a different dot present in the server replica means a conflict.

When a storage system maintains multiple versions, the problems gets more complex. As exemplified in the replica A of Figure 1b, the same strategy can be used to detect concurrent writes from two clients. The problem that arises is what VV to use to identify the second version. When using an entry per server, any VV generated will dominate the VV of the previous version – in the example, $[2,0] < [3,0]$. This can cause problems if it is necessary to compare the two versions, as it would happen in server B, after receiving the version tagged with VV $[3,0]$. This shows that VV with one entry per server are insufficient to track causality among versions generated concurrently by multiple clients.

An alternative used in cloud storage systems, e.g. Riak version , is to keep one entry in the VV per client. This is inefficient as VV can grow very large. To address this problem these systems prune VV optimistically, which is unsafe, possibly leading to lost updates and/or to the introduction of false concurrency. Safe mechanisms for pruning VV, as the one proposed by Golding [1], require global knowledge.

DVV can precisely track causality among versions concurrently created by multiple clients using one entry per replica server. When a client submits a version that is concurrent with the version in the server, a new DVV is generated that correctly tracks causality (as DVV decouple version identification and the causal past). In the example of figure 1c, we have $(A,3)[1,0] \parallel (A,2)[1,0]$.

DVV are a simple, practical and efficient solution to track causality - an evaluation with a modified version of Riak that includes DVV has shown a significant reduction in the size of metadata, and better latency when serving requests [4].

3. RELATED WORK AND CONCLUSIONS

Vector clocks (VC) are used to track causal dependencies among events in a distributed system. The same approach proposed in DVV could be used with VC, as VC use essentially the same mechanism as VV, with the difference that VV only record events that generate new data versions and VC record all events in a distributed system.

Wang et. al. [6] have proposed a variant of VV with $O(1)$ comparison time, but VV entries must be kept ordered, leading to non constant time for other operations. Furthermore, as a simple VV, it also incurs in the problems of VV for tracking causality among concurrent client updates.

WinFS [2] also maintains version identifiers for files separate from the causal past of the whole file system, recorded as a version vector with exceptions (VVE). VVE can express any causal history by recording non-continuous sequences of events.In most multi-version distributed storage systems, a client can only replace all versions in the repository by a new version, making DVV with a single dot sufficient for representing the causal histories that occur. Additionally, WinFS only tracks concurrency among clients running in different nodes, with their own replica and entry in the VVE.

By decoupling the version identifier and the causal past, DVV efficiently record causal dependencies that occurs among clients, allowing to verify causality in $O(1)$ time, instead of $O(n)$ for VV.

Acknowledgments.

We would like to thank Doug Terry for his comments on previous versions of this work. This work is funded by the ERDF, COMPETE Programme,and by National Funds through the FCT, projects FCOMP–01–0124–FEDER–010114, PTDC/EIA-EIA/108963/2008 and PEst-OE/EEI/UI0527/2011.

4. REFERENCES

[1] R. Golding. *Weak-consistency Group Communication and Membership*. PhD thesis, UCSC, 1992.

[2] Dahlia Malkhi and Douglas B. Terry. Concise version vectors in winfs. *Dist. Computing*, 20(3):209–219, 2007.

[3] D. Stott Parker and et. al. Detection of mutual inconsistency in distributed systems. *Trans. on Software Engineering*, 9(3):240–246, 1983.

[4] Nuno Preguiça, Carlos Baquero, Paulo Sérgio Almeida, Victor Fonte, and Ricardo Gonçalves. Dotted version vectors: Logical clocks for optimistic replication. *CoRR*, abs/1011.5808, 2010.

[5] R. Schwarz and F. Mattern. Detecting causal relationships in distributed computations: In search of the holy grail. *Dist. Computing*, 3(7):149–174, 1994.

[6] W. Wang and C. Amza. On optimal concurrency control for optimistic replication. In *Proc. ICDCS*, pages 317–326, 2009.