An empirical perspective on causal consistency

Alejandro Z. Tomsic
Inria Paris-Rocquencourt &
Sorbonne Universités, UPMC Univ Paris 06, LIP6
alejandro.tomsic@lip6.fr

Tyler Crain Marc Shapiro
Inria Paris-Rocquencourt &
Sorbonne Universités, UPMC Univ Paris 06, LIP6
tyler.crain@lip6.fr/marc.shapiro@acm.org

Abstract

Causal consistency is the strongest consistency model under which low-latency and high-availability can be achieved. In the past few years, many causally consistent storage systems have been developed. The long-term goal of this initial work is to perform a deep study and comparison of the different implementations of causal consistency. We identify that protocols that provide causal consistency share the well-known DUR (deferred update replication) algorithmic structure and observe that existing implementations of causal consistency fall into a sub-category of DUR that we name *A-DUR (Asynchronous-DUR)*. In this work, we present the A-DUR algorithmic structure, the pseudocode for the instantiation of two causally consistent protocols under the G-DUR framework, and describe the empirical study we intend to perform on causal consistency.

1. Introduction

The CAP theorem [7] proves that no distributed service can provide strong consistency, availability and partition-tolerance simultaneously; one must be sacrificed. In a distributed setting, network partitions are a given. As a consequence, in the past years, there has been a big amount of research destined to understand the tradeoffs between consistency and availability. Causal consistency has proven to be in the sweetest spot of this tradeoff, i.e., it is the strongest consistency model under which low-latency and high-availability can be achieved [13]. This model is easier to reason about for programmers than eventual consistency, its previously widely-adopted weaker counterpart.

In the past few years, many causally consistent systems have been developed [4–6, 11, 12]. These systems differ in their implementation due to the assumptions and compromises they make. For instance, there are protocols that track potential dependencies [5, 6, 11, 12]; defined by the happens-before relation [10] between events, while others just track explicit dependencies [4, 9]. Another important trade-off is visibility latency vs. throughput [3, 6]. In this line, most protocols use explicit dependency check messages; which improves visibility [1, 4–6, 11, 12] while others improve throughput by utilising a stabilisation mechanism [6] that slightly penalises it.

Choosing among a large number of systems that provide causal consistency can be hard. Even when protocols are well documented, the used vocabulary, naming conventions and perspectives vary. Moreover, design considerations, topology assumptions and implementation differences further constrain the possibility of a fair comparison. Finally, most protocols only compare to a few alternatives and/or to an eventually- or strongly-consistent baseline. It thus remains complicated to understand the important differences among causally-consistent protocols and to make an objective, scientific comparison of their behaviour.

The long-term goal of this initial work is to perform a deep comparative study of the different implementations of causal consistency. As a first-step towards that goal, we need an environment where to implement different protocols. In recent work, Saeida Ardekani et al. [2] identified that there is a family of strongly-consistent protocols that share a generic algorithmic structure, called DUR (deferred update replication). Briefly, DUR protocols execute transactions in two phases: an execution phase, were values are read and updates are buffered; and a termination phase, where an atomic commitment protocol decides on committing or aborting the transaction, and its effects are propagated across the system. Their work presented the G-DUR framework, a tool for implementing DUR protocols, and a deep empirical comparison of relevant strongly consistent systems.

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 609551.

PaPoC'15, April 21, 2015, Bordeaux, France.

http://dx.doi.org/10.1145/2745947.2745949

We identify that causally-consistent protocols also present a DUR structure. Furthermore, we observe that most implementations of causal consistency fall into a sub-category of DUR that we name *A-DUR (Asynchronous-DUR)* and explain in Section 2.

Based on the aforestated, we have decided to implement the protocols for performing our study under G-DUR[2]. In this work, we present the pseudocode for the G-DUR instantiation of two causally consistent protocols[6, 12] (Section 2.1) and describe the empirical study we intend to perform on causal consistency (Section 3).

2. The A-DUR property

A-DUR protocols present particular properties that are not present in every DUR protocol: (i)they are topology (data center) aware; (ii)transactions execute and commit locally: communication only involves local replica(s) of each updated object (normally, the one(s) located at the DC where the transaction is started); (iii)they only incur a termination phase in the case of atomic writes, which never aborts a transaction, and, most distinctively; (iv)they perform *background asynchronous processing*, which handles tasks like propagating committed updates to remote DCs; checking dependencies, resolving conflicts (i.e., causal+ convergence) and applying updates at remote DCs; and/or making updates visible.

2.1 A-DUR protocols under G-DUR

In this section, we present the G-DUR instantiation of two systems that provide causal consistency: GentleRain and Eiger. Namely, the G-DUR instantiation entitles defining the implementations of the following functions: a function θ for partially ordering transactions; $choose$, for choosing a consistent value when reading an object; $certifying\_obj(T_i)$, for determining which objects will be certified during termination phase of a transaction; $commute(T_i, T_j)$, for determining if two transactions T_i and T_j commute; $certify(T_i)$, for deciding if a transaction is safe to be committed; $async\_proc$, for describing the asynchronous processing the protocol performs; and $dep\_check(T_i)$, for checking the dependencies of a transaction T_i at a remote DC. Note that the last two functions are defined specifically for A-DUR instantiations and replace the $post\_commit$ function present in G-DUR. The rest of the code needed to fully implement these protocols is given by the general G-DUR structure [2].

Eiger Eiger[12] (see Algorithm 1) tracks potential dependencies and uses explicit dependency check messages to decide when updates are made visible. Each version of an object stores an identifier composed by a logical timestamp plus a server identifier; and a set of one-hop dependencies which determine causal ordering (line 1). In order to ensure causality, a read operation selects the latest version of an object (Eiger's one round read transaction). When the latest version is not suitable for establishing a causally-consistent snapshot, the read protocol incurs in a second round of reads that selects a version using a timestamp provided by the transaction coordinator (line 2). In order to provide atomic write operations, this protocol relies on a two-phase commit with positive cohorts and indirection (explained elsewhere[12]) that never aborts and only involves the replicas at the local DC (2PC-PCI$_{local}$) holding an object written by the transaction (lines 3-6). After a transaction commits, its effects are sent in the background to the replicas of the updated objects in remote DCs (line 7). At a receiving DC, each server hosting an objected updated by T_i performs a dependency check. It sends messages to the servers at its datacenter in order to check that they have applied the operations identified in each updated object's metadata (line 9).

Algorithm 1 G-DUR instantiation of Eiger

1: $\theta \equiv \{TS\}$
2: $choose \equiv choose_{last\_\{TS\}} \vee choose_{cons}$
3: $\mathcal{AC} \equiv$ 2PC-PCI$_{local}$
4: $certifying\_obj(T_i) \equiv ws(T_i)$
5: $commute(T_i, T_j) \equiv true$
6: $certify(T_i) \equiv true$
7: $async\_proc \equiv send(ws(T_i) \cup \theta(T_i))$
8: to $replicas(ws(T_i)) \setminus local\_replicas(ws(T_i))$
9: $dep\_check(T_i) \equiv \forall\, ts_i \in \theta(T_i)$:
10: $applied_{local}(op_{ts_i.id})$

GentleRain GentleRain[6] (see Algorithm 2) tracks potential dependencies and uses a global stabilisation protocol to make object versions visible. Each version of an object is identified by a physical timestamp, which is used to determine causally consistent snapshots (lines 1 and 2). This protocol does not incur in a termination phase as it does not provide atomic writes. After an object is updated locally, the update is sent in the background to the replicas of the updated objects in remote DCs (line 5). Each server periodically exchanges its local physical clock ($\theta(p)$)with the rest of the servers in the system to compute the GST (global stable time) that is used to make updates visible.

Algorithm 2 G-DUR instantiation of GentleRain

1: $\theta \equiv TS$
2: $choose \equiv choose_{cons}$
3: $certify(T_i) \equiv true$
4: $async\_proc_1 \equiv send(\theta(p))$ to Π
5: $async\_proc_2 \equiv send(ws(T_i) \cup \theta(T_i))$
6: to $replicas(ws(T_i)) \setminus local\_replicas(ws(T_i))$
7: $dep\_check(T_i) \equiv GST < \theta(T_i)$

3. Causal Consistency Study

In this section, we briefly describe the study we intend to realise.

We will start by implementing causally consistent protocols on top of the G-DUR framework, which we expect to save us a significant amount of coding effort. As part of this step, we plan to modify the framework itself in order to optimally support the A-DUR structure. Our study will focus (not exclusively) on a comparison of the realised protocols, an analysis of their bottlenecks and an assessment on the processing and communication costs of causal consistency. In particular, we will analyse the performance of protocols by considering the following tradeoffs and design considerations: potential vs. explicit causality tracking; dependency check messages vs. global stabilisation protocols, provided transactional API, metadata overhead, topology assumptions (partitioning scheme, partial vs. full replication), inter DC replication mechanisms, data staleness and type of causal+ convergence provided (or lack thereof). We plan to run our experiments in Grid'5000 [8], under different configurations and workloads.

With the results obtained from this study, we plan to identify the different flavours of causal consistency and to conclude on the costs of implementing them, when compared to strong and weak consistency.

References

[1] S. Almeida, J. Leitão, and L. Rodrigues. ChainReaction: a causal+ consistent datastore based on Chain Replication. Apr. 2013.

[2] M. S. Ardekani, P. Sutra, and M. Shapiro. G-dur: A middleware for assembling, analyzing, and improving transactional protocols. In *Proceedings of the 15th International Middleware Conference*, Middleware '14, pages 13–24, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2785-5. . URL http://doi.acm.org/10.1145/2663165.2663336.

[3] P. Bailis, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. The potential dangers of causal consistency and an explicit solution. 2012.

[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. ISBN 978-1-4503-2037-5. . URL http://doi.acm.org/10.1145/2463676.2465279.

[5] J. Du, S. Elnikety, A. Roy, and W. Zwaenepoel. Orbe: Scalable causal consistency using dependency matrices and physical clocks. pages 11:1–11:14, Santa Clara, CA, USA, Oct. 2013. . URL http://doi.acm.org/10.1145/2523616.2523628.

[6] J. Du, C. Iorgulescu, A. Roy, and W. Zwaenepoel. Gentlerain: Cheap and scalable causal consistency with physical clocks. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, pages 4:1–4:13, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3252-1. . URL http://doi.acm.org/10.1145/2670979.2670983.

[7] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002. ISSN 0163-5700. .

[8] Grid'5000. Grid'5000, a scientific instrument [...]. https://www.grid5000.fr/, retrieved April 2013.

[9] 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. URL http://dx.doi.org/10.1145/138873.138877.

[10] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978. URL http://doi.acm.org/10.1145/359545.359563.

[11] 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. pages 401–416, Cascais, Portugal, Oct. 2011. .

[12] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Stronger semantics for low-latency geo-replicated storage. pages 313–328, Lombard, IL, USA, Apr. 2013. URL https://www.usenix.org/system/files/conference/nsdi13/nsdi13-final149.pdf.

[13] P. Mahajan, L. Alvisi, and M. Dahlin. Consistency, availability, and convergence. Technical Report UTCS TR-11-22, Dept. of Comp. Sc., The U. of Texas at Austin, Austin, TX, USA, 2011.