Brief Announcement: Byz-GentleRain: An Efficient Byzantine-tolerant Causal Consistency Protocol

Kaile Huang[1], Hengfeng Wei[1⋆], Yu Huang[1], Haixiang Li[2], and Anqun Pan[2]

[1] State Key Laboratory for Novel Software Technology, Nanjing University, China
MG1933024@smail.nju.edu.cn, {hfwei, yuhuang}@nju.edu.cn
[2] Tencent Inc., China
{blueseali, aaronpan}@tencent.com

Abstract. Causal consistency is a widely used weak consistency model and there are plenty of research prototypes and industrial deployments of causally consistent distributed systems. However, none of them consider Byzantine faults, except Byz-RCM proposed by Tseng et al. Byz-RCM achieves causal consistency in the client-server model with $3f+1$ servers where up to f servers may suffer Byzantine faults, but assumes that clients are non-Byzantine. In this work, we present Byz-Gentlerain, the first causal consistency protocol which tolerates up to f Byzantine servers among $3f+1$ servers in each partition and any number of Byzantine clients. Byz-GentleRain is inspired by the stabilization mechanism of GentleRain for causal consistency. To prevent causal violations due to Byzantine faults, Byz-GentleRain relies on PBFT to reach agreement on a sequence of global stable times and updates among servers, and only updates with timestamps less than or equal to such common global stable times are visible to clients. Byz-GentleRain achieves Byz-CC, the causal consistency variant in the presence of Byzantine faults. All reads and updates complete in one round-trip. The preliminary experiments show that Byz-GentleRain is efficient on typical workloads.

Keywords: Causal consistency · Byzantine faults · PBFT · GentleRain

1 Introduction

Causal consistency [1, 2, 6] is a widely used weak consistency model that allows high availability despite network partitions. It guarantees that *an update does not become visible to clients until all its causality are visible.* There are plenty of research prototypes and industrial deployments of causally consistent distributed systems (e.g., COPS [5], GentleRain [4], MongoDB [8], and Byz-RCM [7]). GentleRain in a key-value store uses a stabilization mechanism to make updates

⋆ Corresponding Author. He is also with Software Institute, Nanjing University, China.
This work was partially supported by the CCF-Tencent Open Fund (CCF-Tencent RAGR20200124) and the National Natural Science Foundation of China (61772258).
A full version of this work is available at https://arxiv.org/abs/2109.14189.

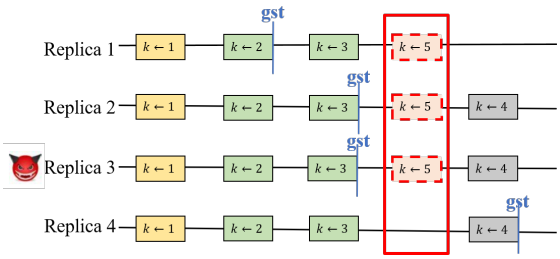

Fig. 1. Why the servers need to synchronize their global stable times.

visible while respecting causal consistency. It timestamps all updates with the physical clock value of the server where they originate. Each server s periodically computes a *global stable time* gst, which is a lower bound on the physical clocks of all servers. This ensures that no updates with timestamps $\leq$ gst will be generated. Thus, it is safe to make the updates with timestamps $\leq$ gst at s visible to clients. A get operation on key k with dependency time dt issued to s will wait until gst $\geq dt$ and then obtain the latest version of k before gst.

However, none of these causal consistency protocols/systems consider Byzantine faults, except Byz-RCM (Byzantine Resilient Causal Memory) in [7]. Byz-RCM achieves causal consistency in the client-server model with $3f + 1$ servers where up to f servers may suffer Byzantine faults, and any number of clients may crash. However, Byz-RCM did not tolerate Byzantine clients, and thus it could rely on clients' requests to identify bogus requests from Byzantine servers [7].

In this work, we present Byz-GentleRain, the first Byzantine-tolerant causal consistency protocol which tolerates up to f Byzantine servers among $3f + 1$ servers in each partition *and* any number of Byzantine clients. It uses PBFT [3] to reach agreement among servers on a total order of client requests. The major challenge Byz-GentleRain faces is to ensure that the agreement is consistent with the causal order. To this end, Byz-GentleRain should prevent causality violations caused by Byzantine clients or servers: Byzantine clients may violate the session order by fooling some servers that a request happened before another that was issued earlier. Byzantine servers may forge causal dependencies by attaching arbitrary metadata for causality tracking to the forward messages. To migrate the potential damages of Byzantine servers, we let clients assign totally ordered timestamps to updates in Byz-GentleRain. Utilizing the digital signatures mechanism, Byzantine servers cannot forge causal dependencies.

To preserve causality, Byz-GentleRain uses the stabilization mechanism of GentleRain. As explained above, the timestamps in Byz-GentleRain are generated by clients. However, it is unrealistic to compute a lower bound on physical clock values of an arbitrary number of clients. Therefore, each server s in Byz-GentleRain maintains and periodically computes a global stable time gst which is a lower bound on physical clock values of the clients it is aware of. Simply refusing any updates with timestamps $\leq$ gst on each server may lead to causality

violations. Consider a system of four servers which are replicas all maintaining a single key k, as shown in Figure 1. Due to asynchrony, these four servers may have different values of gst. Without loss of generality, we assume that $\mathsf{gst}_1 < \mathsf{gst}_2 = \mathsf{gst}_3 < \mathsf{gst}_4$, as indicated by vertical lines. Now suppose that a new update $u : k \leftarrow 5$ with timestamp between gst_3 and gst_4 arrives, and we want to install it on ≥ 3 servers, using quorum mechanism. In this scenario, if each server refuses any updates with timestamps smaller than or equal to its gst, the update u can only be accepted by the first 3 servers, indicated by dashed boxes. Suppose that server 3 is a Byzantine server, which may expose or hide the update u as it will. Consequently, later read operations which read from ≥ 3 servers may or may not see this update u. That is, the Byzantine server 3 may cause causality violations.

To cope with this problem, we synchronize the global stable times of servers. When a server periodically computes its gst, it checks whether no larger global stable time has been or is being synchronized. If so, the server will try to synchronize its gst among all servers, by running PBFT independently in each partition. For each partition, the PBFT leader is also responsible for collecting updates with timestamps $\leq \mathsf{gst}$ from $2f+1$ servers, and synchronizing them on all servers. Once successfully synchronized, a global stable time becomes a *common global stable time*, denoted cgst, and in each partition the updates with timestamps $\leq \mathsf{cgst}$ on all correct servers are the same. Therefore, each server can safely refuse any updates with timestamps smaller than or equal to its cgst.

Still, the classic PBFT is insufficient, since a Byzantine leader of each partition may propose an arbitrary set of updates. To avoid this, the leader will also include the sets of updates it collects from $2f + 1$ servers in its PROPOSE message. A server will reject the PROPOSE message if it finds the contents of this message have been manipulated by checking hash and signatures.

2 Byzantine Causal Consistency

Byz-GentleRain achieves Byzantine Causal Consistency (Byz-CC) defined as follows. For two events e and f, we say that e *happens before* f, denoted $e \rightsquigarrow f$, if and only if one of the following three rules holds:

- *Session-order.* Events e and f are two operation requests issued by the same *correct* client, and e is issued before f.
- *Read-from relation.* Event e is a PUT request issued by some client and f is a GET request issued by a *correct* client, and f reads the value updated by e. Since a GET of Byzantine clients may return an arbitrary value, we do *not* require read-from relation induced by it.
- *Transitivity.* There is another operation request g such that $e \rightsquigarrow g$ and $g \rightsquigarrow f$.

If $e \rightsquigarrow f$, we also say that f causally depends on e and e is a causal dependency of f. A version vv of a key k causally depends on version vv' of key k', if the update of vv causally depends on that of vv'. A key-value store satisfies Byz-CC if, when a certain version of a key is visible to a client, then so are all of its causal dependencies.

3 The Byz-GentleRain Protocol

We consider a distributed multi-version key-value store. It runs at D data centers, each of which has a full copy of a data. In each data center, the full data is shared in to P partitions. We denote by r_d^p the replica of partition p in data center d, and store_d^p the store at replica r_d^p. Each partition consists of at least $3f+1$ replicas and at most f of them may be Byzantine. Any clients may be Byzantine.

When a client issues an update, it assigns to the update a timestamp which is its current clock value. All updates are totally ordered according to their timestamps, with client identifiers used for tie-breaking. We distinguish between the updates that have been received by a server and those that have been made visible to clients. Byz-GentleRain guarantees that an update can be made visible to clients only if so are all its causal dependencies.

In Byz-GentleRain, both clients and servers maintain a *common global stable time* cgst. We denote the cgst at client c by cgst_c and that at replica r_d^p by cgst_d^p. Replicas get their *gst* synchronized using PBFT, to obtain a cgst. All the updates with timestamps $\leq \mathsf{cgst}$ issued to each individual partition will be synchronized as well. Byz-GentleRain maintains the following key invariants about cgst:

- Inv (I): Consider cgst_c at any time σ. All updates issued by correct client c after time σ have a timestamp $> \mathsf{cgst}_c$.
- Inv (II): Consider cgst_d^p at any time σ. No updates with timestamps $\leq \mathsf{cgst}_d^p$ will be successfully executed at $> f$ correct replicas of partition p after time σ.
- Inv (III): Consider a *cgst* value. For any two correct replicas r_d^p and r_i^p (where $i \neq d$) of partition p, if $\mathsf{cgst}_d^p \geq \mathit{cgst}$ and $\mathsf{cgst}_i^p \geq \mathit{cgst}$, then the updates with timestamps $\leq \mathit{cgst}$ in store_d^p and store_i^p are the same.

Each operation returns only when it receives at least $2f+1$ replies from the replicas of the partition it accesses. Byz-GentleRain enforces the following rules for reads and updates:

- Rule (I): For a correct replica r_d^p, any updates with timestamps $> \mathsf{cgst}_d^p$ in store_d^p are invisible to any clients (via GET operations).
- Rule (II): Any correct replica r_d^p will reject any updates with timestamps $\leq \mathsf{cgst}_d^p$.
- Rule (III): For a read operation with timestamp ts issued by client c, any correct replica r_d^p that receives this operation must wait until $\mathsf{cgst}_d^p \geq ts$ before it returns a value to client c.

4 Evaluation

We implement both Byz-GentleRain and Byz-RCM in Java. The key-value stores hold 300 keys in main memory, with each key of size 8 bytes and each value of size 64 bytes. We run all experiments on 4 Aliyun [3] instances running Ubuntu

[3] Alibaba Cloud: `https://www.alibabacloud.com/`.

16.04. Each instance is configured as a data center, with 1 virtual CPU core, 300 MB memory, and 1G SSD storage. All keys are shared into 3 partitions within each data center, according to their hash values.

We first explore the system throughput and the latency of GET and PUT operations of both Byz-GentleRain and Byz-RCM in failure-free scenarios. First, Byz-GentleRain is quite efficient on typical workloads, especially for read-heavy workloads (2,000 $\sim$ 3,000 operations per second). Second, Byz-RCM performs better than Byz-GentleRain, especially with low GET : PUT ratios. This is because Byz-RCM assumes Byzantine fault-free clients and is signature-free. In contrast, Byz-GentleRain requires clients sign each PUT request. Third, the performance of Byz-GentleRain is closely comparable to that of Byz-RCM, if digital signatures are omitted deliberately from Byz-GentleRain.

We then evaluate the impacts of various Byzantine failures on the system throughput of Byz-GentleRain. Specifically, we consider Byzantine clients that may send GET or PUT requests with incorrect timestamps, and Byzantine replicas that may broadcast different global stable time *cgst* to replicas in different partitions. We find that these Byzantine failures have little impact on throughput. In contrast, frequently sending arbitrary messages in PBFT does hurt throughput. This is probably due to the signatures carried by these messages.

References

1. Ahamad, M., Neiger, G., Burns, J.E., Kohli, P., Hutto, P.W.: Causal memory: Definitions, implementation, and programming. Distributed Computing **9**(1), 37–49 (1995). https://doi.org/10.1007/BF01784241
2. Burckhardt, S., Gotsman, A., Yang, H., Zawirski, M.: Replicated data types: Specification, verification, optimality. In: Proceedings of the 41st ACM Symposium on Principles of Programming Languages. pp. 271–284. POPL '14 (2014)
3. Castro, M.: Practical Byzantine fault tolerance. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, USA (2000)
4. Du, J., Iorgulescu, C., Roy, A., Zwaenepoel, W.: Gentlerain: Cheap and scalable causal consistency with physical clocks. In: Proceedings of the ACM Symposium on Cloud Computing. p. 1–13. SoCC '14 (2014)
5. Lloyd, W., Freedman, M.J., Kaminsky, M., Andersen, D.G.: Don't settle for eventual: Scalable causal consistency for wide-area storage with cops. In: Proceedings of the 23rd ACM Symposium on Operating Systems Principles. pp. 401–416. SOSP '11 (2011). https://doi.org/10.1145/2043556.2043593
6. Perrin, M., Mostefaoui, A., Jard, C.: Causal consistency: Beyond memory. In: Proceedings of the 21st ACM Symposium on Principles and Practice of Parallel Programming. PPoPP '16 (2016). https://doi.org/10.1145/2851141.2851170
7. Tseng, L., Wang, Z., Zhao, Y., Pan, H.: Distributed causal memory in the presence of byzantine servers. In: 18th IEEE International Symposium on Network Computing and Applications, NCA 2019. pp. 1–8 (2019)
8. Tyulenev, M., Schwerin, A., Kamsky, A., Tan, R., Cabral, A., Mulrow, J.: Implementation of cluster-wide logical clock and causal consistency in mongodb. In: Proceedings of the 2019 International Conference on Management of Data. pp. 636–650. SIGMOD '19 (2019). https://doi.org/10.1145/3299869.3314049