Brief Announcement: Partially Replicated Causally Consistent Shared Memory^{*†}

Zhuolun Xiang

University of Illinois at Urbana-Champaign
Illinois, USA
xiangzhuolun@gmail.com

Nitin H. Vaidya

University of Illinois at Urbana-Champaign
Illinois, USA
nhv@illinois.edu

ABSTRACT

Distributed shared memory systems maintain multiple replicas of the shared memory registers. Maintaining causal consistency in such systems has received significant attention in the past. However, much of the previous literature focuses on *full* replication wherein each replica stores a copy of all the registers in the shared memory. In this paper, we investigate causal consistency in partially replicated systems, wherein each replica may store only a subset of the shared data. To achieve causal consistency, it is necessary to ensure that, before an update is performed at any given replica, all causally preceding updates must also be performed. Achieving this goal requires some mechanism to track causal dependencies. In the context of full replication, this goal is often achieved using vector timestamps, with the number of vector elements being equal to the number of replicas. Building on the past work, this paper makes two key contributions:

- For a family of algorithms for maintaining causal consistency, we present necessary conditions on the metadata (which we refer as a *timestamp*) that must be maintained by each replica.
- We present an algorithm for achieving causal consistency using a timestamp that matches one of the necessary conditions referred above, thus showing that the condition is necessary and sufficient both.

CCS CONCEPTS

• Theory of computation → Shared memory algorithms;

KEYWORDS

distributed shared memory, causal consistency, partial replication, timestamps, lower bounds

ACM Reference Format:

Zhuolun Xiang and Nitin H. Vaidya. 2018. Brief Announcement: Partially Replicated Causally Consistent Shared Memory. In *PODC '18: ACM Symposium on Principles of Distributed Computing, July 23–27, 2018, Egham, United Kingdom*.

^{*}This research is supported in part by National Science Foundation award 1409416, and Toyota InfoTechnology Center. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the funding agencies or the U.S. government.

[†]A full version of the paper is available in [6].

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

PODC '18, July 23–27, 2018, Egham, United Kingdom

© 2018 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-5795-1/18/07.

<https://doi.org/10.1145/3212734.3212790>

Kingdom. ACM, New York, NY, USA, Article 4, 3 pages. <https://doi.org/10.1145/3212734.3212790>

1 INTRODUCTION

The focus of this work is on causal consistency in a *partially replicated* distributed shared memory (DSM) system that provides the abstraction of shared read/write registers.

This brief announcement focuses on the *peer-to-peer* architecture, where each peer has a *client* that performs read/write operations to the shared memory and a *replica* that helps implement the shared memory abstraction. Extensions to client-server architecture can be found in the full version [6]. We focus on the case when each replica is *partial* and may store a copy of just a subset of the shared registers. *Full replication* is obtained as a special case when each replica stores a copy of each shared register.

For *full replication*, it suffices to use a vector timestamp [4] of length equal to the number of replicas [3] to achieve causal consistency. Several researchers have observed that *partial replication* requires larger amount of metadata to track causal dependencies [1, 2]. Thus, partial replication yields a trade-off between flexibility of replication of the shared registers, and the overhead of the metadata for tracking causality. A goal of our work is to characterize this trade-off. Intuitively, in our solution, each replica maintains an *edge-indexed vector timestamp* which keeps counters for a subset of edges in a “share graph” that characterizes how registers are shared among the replicas. We show that our timestamp is *optimal* in the sense that the subset of share graph edges tracked is necessary for correctness (Theorem 1). Later in Section 3 we present a lower bound for the size of the timestamp space, which matches upper bound achieved by our algorithm in some special cases.

1.1 Preliminaries

We assume an asynchronous system. There are R peers, and hence there are R replicas, numbered 1 through R . Replica i stores copies of a subset of shared registers named X_i . In practice, set X_i for replica i may change dynamically, however, we consider the *static* case in this paper. The client invokes *read* or *write* operations on a shared register stored at the local replica, and waits for a response from the replica. The replicas communicate using reliable point-to-point message-passing channels.

1.2 Algorithm Prototype

We consider causally consistent shared memory that is implemented using an algorithm based on the prototype below, where the structure of the timestamp, how to update the timestamp, and predicate $\mathcal{J}$ are not specified. The choice of the timestamp must ensure that

the timestamp contains adequate information to be able to achieve causal consistency.

Replica prototype: Replica i maintains timestamp τ_i that is suitably initialized.

(1) When replica i receives a $read(x)$ request from the client, replica i responds with the value of the local copy of register x .

(2) When replica i receives a $write(x, v)$ request from the client, replica i performs the following operations atomically:

- (i) write v into the local copy of register x ,
- (ii) update timestamp τ_i as $\tau_i := \text{advance}(i, \tau_i, x, v)$,
- (iii) send $\text{update}(i, \tau_i, x, v)$ message to each other replica $k \in V$ such that $x \in X_k$,
- (iv) return ack to the client.

(3) When replica i receives a message $\text{update}(k, \tau_k, x, v)$, it adds $\text{update}(k, \tau_k, x, v)$ to a local data structure named pending_i .

(4) For any $\text{update}(k, \tau_k, x, v) \in \text{pending}_i$, when predicate $\mathcal{J}(i, \tau_i, k, \tau_k)$ evaluates True, replica i performs the following operations atomically:

- (i) writes value v to its local copy of register x ,
- (ii) updates its timestamp as $\tau_i := \text{merge}(i, \tau_i, k, \tau_k)$,
- (iii) removes $\text{update}(k, \tau_k, x, v)$ from pending_i .

Notation for updates: We will use short labels, such as u or u_1 , to represent an update. When all operations in step 2, or all operations in step 4 have been performed, we say that $\text{update}(i, \tau, x, v)$ has been *applied* at replica i . The replica i is said to *issue an update* when step 2 is performed.

1.3 Replica-centric Causal Consistency

The causal consistency model addressed in our work is inspired by replicated shared memory systems such as Lazy Replication [3]. We refer to this model as the *replica-centric* causal consistency model.

DEFINITION 1 (HAPPENED-BEFORE RELATION $\hookrightarrow$ FOR UPDATES). Given updates u_1 and u_2 , $u_1 \hookrightarrow u_2$ if and only if at least one of the following conditions is true:

- (i) u_1 is applied at a replica sometime before the same replica issues u_2 ,
- (ii) There exists an update u_3 such that $u_1 \hookrightarrow u_3$ and $u_3 \hookrightarrow u_2$.

Intuitively, an update issued by replica i is considered causally dependent on any updates that were previously applied at that replica, regardless of whether the previously updated registers were read by the client or not. We define *replica-centric* causal consistency more formally now using relation $\hookrightarrow$.

DEFINITION 2. Replica-centric causal consistency is achieved if the following two properties are satisfied:

- **Safety:** If an update u_1 for register $x \in X_i$ has been applied at a replica i , then there must not exist update u_2 for some register in X_i such that (i) $u_2 \hookrightarrow u_1$, and (ii) replica i has not yet applied u_2 .

- **Liveness:** Any update u issued by a replica i for a register $x \in X_i$ will eventually be applied at each replica j such that $x \in X_j$.

For two important reasons, we consider the replica-centric causal consistency in this paper. (i) First, the lower bounds on timestamp size presented in Section 3 for the replica-centric causal consistency is also a lower bound for the client-centric causal consistency. (ii) Second, in practice, the replica-centric approach is convenient, since it only uses a single timestamp per replica (as compared to, for instance, a timestamp per register per replica). Many practical systems, including Lazy Replication [3] and SwiftCloud [7], in fact, conform to the replica-centric view.

2 TIMESTAMPS FOR REPLICA-CENTRIC CAUSAL CONSISTENCY

In this section, we identify a necessary and sufficient condition on the timestamp τ_i maintained by each replica i in the prototype in order to achieve the replica-centric causal consistency. Intuitively, our condition identifies a subset of directed edges in the share graph that are *necessary and sufficient* to “keep track” of for each replica in order to achieve replica-centric causal consistency.

Hélary and Milani [2] introduced the notion of a *share graph* to represent a partially replicated system. We define $X_{ij} = X_i \cap X_j$, the set of registers stored at replicas i and j both. In the following, e_{ij} is a directed edge from i to j .

DEFINITION 3 (Share Graph [2]). Share graph is defined as a directed graph $G = (V, E)$, where $V = \{1, 2, \dots, R\}$, and vertex $i \in V$ represents replica i . There exist directed edges e_{ij} and e_{ji} in E if and only if $X_{ij} \neq \emptyset$.

For a replica i , and directed edge e_{jk} (from j to k), Definition 4 defines an (i, e_{jk}) -loop.

DEFINITION 4 ((i, e_{jk})-loop). Given replica i and edge e_{jk} ($j \neq i \neq k$) in share graph G , consider a simple loop of the form $(i, l_1, l_2, \dots, l_s = k, j = r_1, r_2, \dots, r_t, i)$, where $s \geq 1$ and $t \geq 1$. Define $i = r_{t+1}$. The simple loop is said to be an (i, e_{jk}) -loop provided that:

- (i) $X_{jk} - \left(\bigcup_{1 \leq p \leq s-1} X_{l_p} \right) \neq \emptyset$,
- (ii) $X_{jr_2} - \left(\bigcup_{1 \leq p \leq s-1} X_{l_p} \right) \neq \emptyset$, and
- (iii) for $2 \leq q \leq t$, $X_{r_q r_{q+1}} - \left(\bigcup_{1 \leq p \leq s} X_{l_p} \right) \neq \emptyset$.

To help present the necessary condition in Section 2.1, we now define a *timestamp graph*. Intuitively, timestamp graph G_i consists of directed edges that are *necessary and sufficient* for replica i to keep track of in its timestamp, as we will show later in this section.

DEFINITION 5 (Timestamp graph G_i of replica i). Given share graph $G = (V, E)$, timestamp graph of replica i is defined as a directed graph $G_i = (V_i, E_i)$, where

$$\begin{aligned} E_i &= \{e_{ij} \mid e_{ij} \in E\} \cup \{e_{ji} \mid e_{ji} \in E\} \cup \\ &\quad \{e_{jk} \mid \exists (i, e_{jk})\text{-loop in } G, j \neq i \neq k, e_{jk} \in E\} \\ V_i &= \{u, v \mid e_{uv} \in E_i\} \end{aligned}$$

Thus, E_i consists of all directed edges incident at i , and each edge $e_{jk} \in E$ such that an (i, e_{jk}) -loop exists. Note that timestamp edges are not necessarily bidirectional.

2.1 A Necessary Condition for Timestamps

In order to achieve replica-centric causal consistency, the timestamp maintained by each replica must contain adequate information, as shown in Theorem 1.

THEOREM 1 (INFORMAL). *Consider a shared memory system that implements replica-centric causal consistency using an algorithm conforming with the prototype in Section 1.2. Any replica i 's timestamp has to be dependent on the updates performed on edge e_{jk} for each $e_{jk} \in E_i$.*

Formal statement of the above theorem can be found in [6]. In Section 2.2 we present an algorithm that uses timestamps precisely satisfy the necessary condition above, proving that the necessary condition in Theorem 1 is sufficient as well.

2.2 Sufficiency of Tracking Edges in Timestamp Graph

The algorithm uses an *edge-indexed* vector timestamp. In particular, replica i 's timestamp τ_i is a vector of length $|E_i|$ with its elements being indexed using edges in E_i . Recall that timestamp graph edges sets E_i and E_j may be distinct for distinct replicas i and j . Therefore, the timestamps maintained by different replicas may have different sizes, and indexed by different edges. Despite this non-uniformity, the timestamps suffice to implement replica-centric causal consistency.

Proposed algorithm: Since the algorithm conforms to prototype in Section 1.2, it suffices to describe the timestamp structure, *advance* and *merge* functions, and predicate $\mathcal{J}$ for each replica i .

- Timestamp maintained by replica i : Each replica i maintains a vector timestamp τ_i that is indexed by the edges in E_i . For edge $e_{jk} \in E_i$, $\tau_i[e_{jk}]$ is an integer, initialized to 0.
- Function *advance*(i, τ_i, x, v) returns vector T_i (indexed by edges in E_i) defined as follows. For each $e_{jk} \in E_i$:

$$T_i[e_{jk}] := \begin{cases} \tau_i[e_{jk}] + 1, & \text{if } j = i \text{ and } x \in X_{ik}, \\ \tau_i[e_{jk}], & \text{otherwise} \end{cases}$$

Observe that *advance*(i, τ_i, x, v) increments elements of τ_i corresponding to edges from i to only those replicas that also store register x .

- Function *merge*(i, τ_i, k, τ_k) returns following vector T_i (indexed by edges in E_i):

$$T_i[e] := \begin{cases} \max(\tau_i[e], \tau_k[e]), & \text{for each edge } e \in E_i \cap E_k, \\ \tau_i[e], & \text{for each edge } e \in E_i - E_k \end{cases}$$

- For $k \neq i$, predicate $\mathcal{J}(i, \tau_i, k, \tau_k) = \text{True}$ if and only if $\tau_i[e_{ki}] = \tau_k[e_{ki}] - 1$ and $\tau_i[e_{ji}] \geq \tau_k[e_{ji}]$, for each $e_{ji} \in E_i \cap E_k$, $j \neq k$.

The full version [6] presents several optimizations that can reduce the timestamps sizes in practice.

3 LOWER BOUND ON TIMESTAMP SIZE

In this section, we present a lower bound (in bits) on the size of the timestamps for the restricted case where timestamps are assigned as a function of the *causal past* of a replica.

DEFINITION 6 (Causal past[5]). *Causal past S of a replica that has applied updates in set U consists of updates in*

$$S = U \cup \{u' \mid u \in U, u' \hookrightarrow u\}$$

Recall that $G = (V, E)$ denotes the share graph. For $e_{jk} \in E$, let $S|_{e_{jk}}$ denote the set of updates in S that are issued by replica j on registers in X_{jk} . For $e_{jk} \notin E$, define $S|_{e_{jk}} = \emptyset$ for convenience.

DEFINITION 7 (CONFLICT). *Given share graph $G = (V, E)$, and two possible causal pasts S_1, S_2 of replica i from executions where each replica issues up to m updates, we say that S_1 and S_2 conflict if following conditions hold:*

- (1) $\forall e \in E, S_1|_e \neq \emptyset \neq S_2|_e$, and
- (2) $\exists e \in E$ such that $S_1|_e \subset S_2|_e$, where $e = e_{ij}$ or $e = e_{ji}$ or $\exists$ a simple loop $(i, l_1, \dots, l_s, r_1, \dots, r_t, i = r_{t+1}) \in G$ where $e = e_{r_1 l_s}$ such that
 - (1) $S_1|_{e_{r_p l_q}} = S_2|_{e_{r_p l_q}}$ for $1 \leq p \leq t+1, 1 \leq q \leq s$ and $e_{r_p l_q} \neq e_{r_1 l_s}$, and
 - (2) $S_x|_{e_{r_p r_{p+1}}} - \bigcup_{1 \leq q \leq s} S_x|_{e_{r_p l_q}} \neq \emptyset$ for $1 \leq p \leq t$ and $x = 1, 2$

LEMMA 1. *Consider two possible causal pasts S_1, S_2 of replica i . If S_1 and S_2 conflict, then distinct timestamps must be assigned to them for ensuring the safety and liveness properties in Definition 2.*

For replica i , we define a **conflict graph** H_i with vertex set equal to the set of all possible causal pasts of replica i . An edge is added between any two causal pasts of replica i that conflict. Then, the *chromatic number* for the conflict graph is a lower bound on timestamp space size. Therefore, we have the following theorem.

THEOREM 2. *Consider a shared memory system that implements replica-centric causal consistency using an algorithm conforming with the prototype in Section 1.2. Let $\chi(H_i)$ denote the chromatic number of conflict graph H_i . Then, for partial replication, the timestamp space size for any replica i is lower bounded by $\chi(H_i)$.*

Implication: It can be shown that when the share graph is a tree, a cycle of n replicas, or a clique representing full replication, the above theorem implies a lower bound on the timestamp size that is *tight* due to the fact that our algorithm also uses timestamp of the same size.

Summary: This work investigates partially replicated causally consistent shared memory systems. We present some necessary conditions and an algorithm that achieves replica-centric causal consistency in a partially replicated system.

REFERENCES

- [1] Tyler Crain and Marc Shapiro. 2015. Designing a causally consistent protocol for geo-distributed partial replication. In *Proceedings of the First Workshop on Principles and Practice of Consistency for Distributed Data*. ACM.
- [2] Jean-Michel Hélary and Alessia Milani. 2006. About the efficiency of partial replication to implement distributed shared memory. In *International Conference on Parallel Processing*. IEEE.
- [3] Rivka Ladin, Barbara Liskov, Liuba Shrira, and Sanjay Ghemawat. 1992. Providing high availability using lazy replication. *ACM Transactions on Computer Systems (TOCS)* (1992).
- [4] Friedemann Mattern. 1988. Virtual Time and Global States of Distributed Systems. In *Workshop on Parallel and Distributed Algorithms*.
- [5] Giovanna Melideo. 2001. *Tracking Causality in Distributed Computations*. Ph.D. Dissertation.
- [6] Zhuolun Xiang and Nitin H Vaidya. 2017. Lower Bounds and Algorithm for Partially Replicated Causally Consistent Shared Memory. *arXiv preprint arXiv:1703.05424* (2017).
- [7] Marek Zawirski, Annette Bieniusa, Valter Bolegas, Sérgio Duarte, Carlos Baquero, Marc Shapiro, and Nuno Preguiça. 2013. SwiftCloud: Fault-tolerant geo-replication integrated all the way to the client machine. *arXiv preprint arXiv:1310.3107* (2013).