Brief Announcement: Optimal Record and Replay under Causal Consistency*

Russell L. Jones
University of Illinois at
Urbana-Champaign
rjones27@illinois.edu

Muhammad S. Khan
University of Illinois at
Urbana-Champaign
mskhan6@illinois.edu

Nitin H. Vaidya
University of Illinois at
Urbana-Champaign
nhv@illinois.edu

ABSTRACT

We investigate the minimum record needed to replay executions of processes that share causally consistent memory. For a version of causal consistency, we identify optimal records under both offline and online recording setting. Under the offline setting, a central authority has information about every process' view of the execution and can decide what information to record for each process. Under the online setting, each process has to decide on the record at runtime as the operations are observed.

CCS CONCEPTS

• **Theory of computation** → **Shared memory algorithms**;

ACM Reference Format:

Russell L. Jones, Muhammad S. Khan, and Nitin H. Vaidya. 2018. Brief Announcement: Optimal Record and Replay under Causal Consistency. In *PODC '18: ACM Symposium on Principles of Distributed Computing, July 23–27, 2018, Egham, United Kingdom*. ACM, New York, NY, USA, 3 pages. <https://doi.org/10.1145/3212734.3212789>

1 INTRODUCTION

In this paper, we explore *Record and Replay* (RnR) of multi-process applications where processes communicate via shared memory. Record and replay mechanisms aim to allow parallel program debugging to proceed as follows. The programmer runs the program, and potentially observes incorrect behavior. The programmer then reruns the program, while more closely watching the program state, and attempts to discover where a program bug may have occurred. However, even when a parallel program is re-executed with the same input, different executions of the program may proceed differently, due to non-determinism introduced by the uncertainty in the delays incurred in performing various operations. Thus, the observed bug may not re-occur during rerun, making it quite difficult to discover the cause of the original problem. Record and Replay (RnR) aims to solve this problem by creating a *record* during the original execution, and using it during replay to guarantee that the rerun produces the same outcome as the original execution. In other

words, while the original execution may be non-deterministic, the replay using the record eliminates the non-determinism as desired.

This work is motivated by the trade-off between the consistency model for shared memory and the amount of information that must be recorded to facilitate a replay. A stronger consistency model imposes more constraints on the execution, resulting in a smaller space of allowed executions. Intuitively, a stronger consistency model should require a smaller record to resolve the non-determinism during replay. In the full paper [4] we present a concrete example execution to illustrate that this intuition is indeed correct. In prior work, Netzer [7] identified the minimum record necessary for RnR under the *sequential consistency* model. The computer architecture research community has also investigated RnR systems under various consistency models (e.g., [6], [1], [2]). However, to the best of our knowledge, only Netzer's work [7] has addressed identification of minimum record for RnR.

This paper builds on Netzer's work to address the minimum record for correct replay under *causal consistency*. Whether a certain record is necessary and sufficient for replay depends on several factors, as discussed in the full paper [4]. We describe the RnR model for this work in Section 4.

2 PRELIMINARIES

A *relation* R on a set O is a set of tuples (a, b) such that $a, b \in O$. We use the notation $a <_R b$ if $(a, b) \in R$. We denote $a \leq_R b$ if either $a <_R b$ or $a = b$. An irreflexive, antisymmetric, and transitive relation is called a *partial order*. A partial order R on a set O is a *total order* if for any $a, b \in O$, either $a <_R b$ or $b <_R a$. A partial order can be represented by a directed acyclic graph which is closed under transitivity. For two partial orders A and B on a set O , we say that A *respects* B if $B \subseteq A$. We use the notation $A \upharpoonright O'$ to restrict the partial order A on set O to a subset $O' \subseteq O$. $\hat{A}$ denotes the (unique) transitive reduction of A and $a <_A b$ denotes $(a, b) \in \hat{A}$. We use $A \cup B$ to denote the union, with the transitive closure, of A and B , and $A \dot{\cup} B$ to denote the disjoint union of A and B . For example, consider two partial orders A and B on the set $\{a, b\}$, given by $A = \{(a, b)\}$ and $B = \{(b, a)\}$. Then, $A \cup B = \{(a, b), (b, a), (a, a), (b, b)\}$ while $A \dot{\cup} B = \{(a, b), (b, a)\}$. Observe that union and disjoint union of two partial orders may not be a partial order, as the previous example shows.

We borrow some notation by Steinke and Nutt [8] for shared memory formalism. The shared memory consists of a set of variables X and supports two operations, read and write. We use w for writes, r for reads, and o when the operation can be either read or write. We use a subscript for process identifier or leave it blank if it is unspecified. Formally, an *operation* is a 4-tuple (op, i, x, id) where op is r for read and w for write, i is the unique identifier of the

*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.

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/10.1145/3212734.3212789>

process that performed the operation, x is the (shared) variable on which the operation was performed, and id is the unique identifier of the operation. This notation allows for wild-card entries, e.g. $(w, i, *, *)$ is the set of all writes executed by process i . Observe that we do not specify the values in the notation. We assume that each write operation writes a unique value¹. The values read by read operations may vary between executions, but each read operation reads a value written by some write.

All operations in $(*, i, *, *)$ are totally ordered. We denote this total order by $PO(i)$. The *program order* is given by $PO = \cup_i PO(i)$. We assume that the program order PO is fixed. Formally, a shared memory system is a set of processes P , a set of operations O , a program order PO on O , a set of shared variables X , and a shared memory Π . An execution is the result of processes running their programs on a shared memory system where each read operation returns a value written by some write operation.

Definition 2.1 (Writes-to). Given an execution, a write operation w writes-to a read operation r , denoted $w \mapsto r$, if w and r are on the same variable and r returns the value written by w .

3 SHARED MEMORY CONSISTENCY

For an execution, a *view* V on a set of operations $O' \subseteq O$ is a total order on O' such that each read $r \in O'$ returns the last value written to the corresponding variable in V . Reasoning about allowed executions under a shared memory consistency model relies on existence of some collection of views $\mathcal{V}$ that satisfy some properties, depending on the shared memory consistency model. We say that $\mathcal{V}$ *explains* the execution under the consistency model. For example, causal consistency requires existence of per-process views that satisfy causality, which is the union (with the transitive closure) of the writes-to relation and the program order. Formally, we use the definition by Steinke and Nutt [8].

Definition 3.1 (Write-read-write Order [Steinke and Nutt [8]]). Given an execution with a writes-to relation $\mapsto$, two writes, $w^1 \in (w, *, *, *)$ and $w^2 \in (w, *, *, *)$, are ordered by *write-read-write order*, $(w^1, w^2) \in WO$, if there exists a read operation $r \in (r, *, *, *)$ such that $w^1 \mapsto r <_{PO} w^2$.

Definition 3.2 (Causal Consistency [Steinke and Nutt [8]]). An execution is causally consistent if there exists a set of views $\mathcal{V} = \{V_i\}_{i \in P}$ such that, for every process i ,

- V_i is a view on the set of operations $(*, i, *, *) \cup (w, *, *, *)$, and
- V_i respects $WO \cup (PO|(*, i, *, *) \cup (w, *, *, *))$.

A shared memory Π is *causally consistent* if every execution run on Π is causally consistent.

Note that, by definition, each view V_i already respects the writes-to relation restricted to $(*, i, *, *) \cup (w, *, *, *)$. Note also that read operations are only observed by the processes that perform them while write operations are observed by every process. We work with a version of causal consistency which we call *strong causal consistency*. This model is motivated by an implementation of causal consistency via lazy replication [5]. Ladin et. al. [5] use vector timestamps to ensure that a write operation w_i from process i is only

¹The unique write values have a one-to-one correspondence with the unique identifiers of the respective write operations.

committed locally when all write operations in w_i 's history, as summarized by w_i 's vector timestamp, have been observed. Formally, we define strong causal consistency as follows.

Definition 3.3 (Strong Causal Order). Given a set of views $\mathcal{V} = \{V_i\}_{i \in P}$, two writes, $w^1 \in (w, *, *, *)$ and $w^2 \in (w, i, *, *)$, are ordered by *strong causal order*, $(w^1, w^2) \in SCO(\mathcal{V})$, if $(w^1, w^2) \in V_i$.

Definition 3.4 (Strong Causal Consistency). An execution is *strongly causal consistent* if there exists a set of views $\mathcal{V} = \{V_i\}_{i \in P}$ such that, for every process i ,

- V_i is a view on the set of operations $(*, i, *, *) \cup (w, *, *, *)$, and
- V_i respects $SCO(\mathcal{V}) \cup (PO|(*, i, *, *) \cup (w, *, *, *))$.

A shared memory Π is *strongly causally consistent* if every execution run on Π is strongly causally consistent.

Observe that strong causal consistency does not violate the write-read-write order and thus it is at least as strong as causal consistency. In fact, it is strictly stronger than causal consistency, as shown in the full paper [4]. In this work we focus on optimal record for strong causal consistency.

4 RNR MODEL

For replaying executions, we assume that the per-process views are provided to the RnR system. The RnR system uses the views to determine the record. In case of online recording, the views are provided to the RnR system incrementally, as and when new operations occur that affect the views. Now let us illustrate how this requirement may be implemented in practice. Consider a shared memory implementation wherein each process has a copy of the shared variable and the shared memory is implemented via message passing. Then the shared memory adds a write operation to a process i 's view when the local copy of the corresponding variable is updated at process i . Similarly a read by process i is added to process i 's view when the local copy is read.

The RnR system will record some edges from each view (i.e. on each process) and the replay execution is only allowed views that enforce these records. Note that we do not place any restriction on how the record is enforced. We assume that any set of views can explain the replay as long as it extends the record and is consistent under the shared memory consistency model. Formally, given a set of views $\mathcal{V} = \{V_i\}_{i \in P}$, $\mathcal{R} = \{R_i\}_{i \in P}$ is a *record* of $\mathcal{V}$ if each $R_i \subseteq V_i$. An execution is a *replay* of $\mathcal{R}$ if there exists a set of views $\mathcal{V}' = \{V'_i\}_{i \in P}$ that explain the execution under the consistency model and each V'_i respects R_i . We say that $\mathcal{V}'$ *certifies* the replay to be valid for $\mathcal{R}$. A record $\mathcal{R}$ of a set of views $\mathcal{V}$ is *good* if, for any replay of $\mathcal{R}$, under the same consistency model, any set of views $\mathcal{V}' = \{V'_i\}_{i \in P}$ that certifies the replay to be valid for $\mathcal{R}$ must have $V'_i = V_i$ for all $i \in P$ (i.e. only $\mathcal{V}$ certifies the replay to be valid for $\mathcal{R}$). Note that for each record, there exists at least one replay, specifically the original execution.

5 OPTIMAL RECORD FOR STRONG CAUSAL CONSISTENCY

Offline Record

In this section we consider offline record for strong causal consistency. In this case the entire set of per-process views $\mathcal{V} = \{V_i\}_{i \in P}$

are made available to the RnR system. The RnR system determines the record that must be saved. We first present what edges do not need to be recorded. Consider a process i . Since PO is fixed and independent of executions the RnR system does not have to record these edges in V_i as they are guaranteed by the consistency model. Now consider two write operations $w^1 \in (w, *, *, *)$ and $w_j^2 \in (w, j, *, *)$, for $j \neq i$, such that $(w^1, w_j^2) \in SCO(\mathcal{V})$. If process j correctly orders the two operations (w^1, w_j^2) in the replay, then this edge will be guaranteed by the consistency model, due to strong causal order, and process i does not need to record it. Such edges are captured by the following definition.

Definition 5.1. Given a set of views $\mathcal{V} = \{V_i\}_{i \in P}$, the relation $SCO_i(\mathcal{V})$, for a process $i \in P$, is defined as follows. Two writes, $w^1 \in (w, *, *, *)$ and $w_j^2 \in (w, j, *, *)$, are ordered $(w^1, w_j^2) \in SCO_i(\mathcal{V})$, if $(w^1, w_j^2) \in SCO(\mathcal{V})$ and $j \neq i$.

The following set of edges also do not need to be recorded.

Definition 5.2. Given a set of views $\mathcal{V} = \{V_i\}_{i \in P}$, the relation $B_i(\mathcal{V})$, for a process $i \in P$, is defined as follows. Two writes, $w_i^1 \in (w, i, *, *)$ and $w_j^2 \in (w, j, *, *)$ such that $i \neq j$, are ordered $(w_i^1, w_j^2) \in B_i(\mathcal{V})$ if $(w_i^1, w_j^2) \in V_i$ and there exists a process $k \neq i, j$ such that $(w_i^1, w_j^2) \in V_k$.

In any set of views $\mathcal{V}'$ that explain a replay of $\mathcal{R}$, setting $(w_j^2, w_i^1) \in V'_i$ will create an $SCO(\mathcal{V}')$ edge (w_j^2, w_i^1) which will conflict with V'_k . Theorems 5.3 and 5.4 show that the record $\mathcal{R} = \{R_i\}_{i \in P}$ such that $R_i = \widehat{V}_i \setminus (SCO_i(\mathcal{V}) \cup PO \cup B_i(\mathcal{V}))$ is both sufficient and necessary for a correct replay under strong causal consistency.

THEOREM 5.3. Given a set of views $\mathcal{V} = \{V_i\}_{i \in P}$ that explain a strongly causal consistent execution. For each process $i \in P$, let $R_i = \widehat{V}_i \setminus (SCO_i(\mathcal{V}) \cup PO \cup B_i(\mathcal{V}))$. Then, $\mathcal{R} = \{R_i\}_{i \in P}$ is a good **online** record of $\mathcal{V}$.

THEOREM 5.4. Given a set of views $\mathcal{V} = \{V_i\}_{i \in P}$ that explain a strongly causal consistent execution. For any good record $\mathcal{R} = \{R_i\}_{i \in P}$ of $\mathcal{V}$, for any process $i \in P$ and any two operations $o^1, o^2 \in (*, i, *, *) \cup (w, *, *, *)$, if $(o^1, o^2) \in \widehat{V}_i \setminus PO \cup SCO_i(\mathcal{V}) \cup B_i(\mathcal{V})$, then $(o^1, o^2) \in R_i$.

Online Record

In the online setting each process has a fixed set of read and write operations $(*, i, *, *)$ that it executes in their local order $PO(i)$ by communicating with the shared memory. Executing an operation may take arbitrarily long and the process may spend arbitrarily long time to execute the next operation but each process only executes one operation at a time. Via the shared memory, a process i observes its own operations and write operations from other processes one at a time. The order in which these operations are observed give rise to the view V_i . More formally, the execution proceeds in time steps. At each time step in the execution, a unique² process i observes an operation from $(*, i, *, *) \cup (w, *, *, *)$ and adds it to its view V_i .

The online record algorithm proceeds as follows. Suppose process i wants to record $(o^1, o^2) \in V_i$. Then, process i must record

²Uniqueness of the process makes the model simpler. If more than one process observes an operation at a given time step, we can separate this into multiple time steps ordered by the process identifiers.

(o^1, o^2) at the time when it observes o^2 . In the online setting, process i has limited information about views of other processes at any given time in the execution. How much does process i know? We assume that, at most, process i has access to the history of other processes brought with the observed operation. More precisely, at any time in the execution, if process i is *aware* that $(o^1, o^2) \in V_j$, for some process $j \neq i$, then process i must have already observed $o_j^3 \in (*, j, *, *)$ such that $o^1 <_{V_j} o^2 \leq_{V_j} o_j^3$. As discussed in Section 4, the recording proceeds without information about the internal workings of the shared memory. However, we assume that the RnR system is aware of the shared memory guarantees. More precisely, for strong causal consistency, we assume that any process i can check if $(o^1, o^2) \in SCO(\mathcal{V})$ and also if $(o^1, o^2) \in PO$. For a given execution $\mathcal{V} = \{V_i\}_{i \in P}$, we say that a record $\mathcal{R} = \{R_i\}_{i \in P}$ is an **online record** of $\mathcal{V}$ if $\mathcal{R}$ can be recorded in this manner.

Theorems 5.5 and 5.6 state that for each process i , $R_i = \widehat{V}_i \setminus (SCO_i(\mathcal{V}) \cup PO)$ is both sufficient and necessary in the online setting.

THEOREM 5.5. Given a set of views $\mathcal{V} = \{V_i\}_{i \in P}$ that explain a strongly causal consistent execution. For each process $i \in P$, let $R_i = \widehat{V}_i \setminus (SCO_i(\mathcal{V}) \cup PO)$. Then, $\mathcal{R} = \{R_i\}_{i \in P}$ is a good **online** record of $\mathcal{V}$.

THEOREM 5.6. Given a set of views $\mathcal{V} = \{V_i\}_{i \in P}$ that explain a strongly causal consistent execution. For any good **online** record $\mathcal{R} = \{R_i\}_{i \in P}$ of $\mathcal{V}$, for any process $i \in P$ and any two operations $o^1, o^2 \in (*, i, *, *) \cup (w, *, *, *)$, if $(o^1, o^2) \in \widehat{V}_i \setminus PO \cup SCO_i(\mathcal{V})$, then $(o^1, o^2) \in R_i$.

SUMMARY

In this brief announcement we have stated our results for the optimal record for RnR under strong causal consistency. The full paper [4] contains more details and proofs. The optimal record for causal consistency is still an open problem.

REFERENCES

- [1] George W. Dunlap, Dominic G. Lucchetti, Michael A. Fetterman, and Peter M. Chen. 2008. Execution Replay of Multiprocessor Virtual Machines. In *Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '08)*. ACM, New York, NY, USA, 121–130. <https://doi.org/10.1145/1346256.1346273>
- [2] Nima Honarmand and Josep Torrellas. 2014. RelaxReplay: Record and Replay for Relaxed-consistency Multiprocessors. *SIGARCH Comput. Archit. News* 42, 1 (Feb. 2014), 223–238. <https://doi.org/10.1145/2654822.2541979>
- [3] Russell Llewellyn Jones. 2016. *Record and Replay under Relaxed Consistency*. Master's thesis. University of Illinois at Urbana-Champaign. <http://hdl.handle.net/2142/92729>
- [4] Russell L. Jones, Muhammad S. Khan, and Nitin H. Vaidya. 2018. Optimal Record and Replay under Causal Consistency. In *arXiv: to be submitted*.
- [5] Rivka Ladin, Barbara Liskov, Liuba Shrira, and Sanjay Ghemawat. 1992. Providing High Availability Using Lazy Replication. *ACM Trans. Comput. Syst.* 10, 4 (Nov. 1992), 360–391. <https://doi.org/10.1145/138873.138877>
- [6] Dongyoon Lee, Benjamin Wester, Kaushik Veeraraghavan, Satish Narayanasamy, Peter M. Chen, and Jason Flinn. 2010. Respec: Efficient Online Multiprocessor Replay via Speculation and External Determinism. *SIGPLAN Not.* 45, 3 (March 2010), 77–90. <https://doi.org/10.1145/1735971.1736031>
- [7] Robert H. B. Netzer. 1993. Optimal Tracing and Replay for Debugging Shared-memory Parallel Programs. *SIGPLAN Not.* 28, 12 (Dec. 1993), 1–11. <https://doi.org/10.1145/174267.174268>
- [8] Robert C. Steinke and Gary J. Nutt. 2004. A Unified Theory of Shared Memory Consistency. *J. ACM* 51, 5 (Sept. 2004), 800–849. <https://doi.org/10.1145/1017460.1017464>