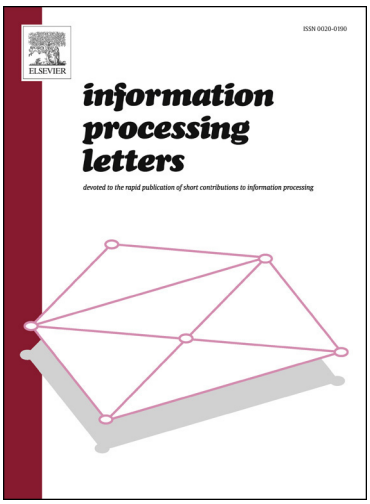
Accepted Manuscript



Analyzing linearizability violations in the presence of read-modify-write operations

H. Fan, W. Golab

PII: S0020-0190(18)30129-7
DOI: https://doi.org/10.1016/j.ipl.2018.06.004
Reference: IPL 5706

To appear in: *Information Processing Letters*

Received date: 30 August 2014
Revised date: 23 November 2017
Accepted date: 5 June 2018

Please cite this article in press as: H. Fan, W. Golab, Analyzing linearizability violations in the presence of read-modify-write operations, *Inf. Process. Lett.* (2018), https://doi.org/10.1016/j.ipl.2018.06.004

Highlights

- We focus on “black-box” testing of the consistency of distributed system.
- It uses as input histories of read, write and RMW operations invoked by clients.
- We present a novel algorithm for computing Γ metric of inconsistency in histories.
- We characterize linearizability violation to handle read-modify-write operations.

Analyzing Linearizability Violations in the Presence of Read-Modify-Write Operations

H. Fan[a,*], W. Golab[a,1,**]

[a] *Department of Electrical and Computer Engineering, University of Waterloo*
200 University Ave. West, Waterloo, Ontario, N2L 3G1, Canada

Abstract

We consider an algorithmic problem related to analyzing consistency anomalies in distributed storage systems. Specifically, given a history of read, write, and read-modify-write operations applied by clients, we quantify how far the history deviates from the "gold standard" of *linearizability* (Herlihy and Wing, 1990). Our solution generalizes a known algorithm that considers reads and writes only.

Keywords: distributed storage, consistency, linearizability, verification

1. Introduction

Distributed storage systems support essential online services including web search, social networking, and cloud file sharing. To meet stringent demands for high availability and low latency, such systems maintain multiple replicas of data, often in geographically distributed data centers. Storage operations are therefore executed using distributed protocols that ensure crucial correctness properties in a highly concurrent, failure-prone environment. Reasoning about the correctness of such protocols is notoriously difficult, especially for systems that support lightweight transactions (e.g., conditional write operations, increments) and latency-reducing optimizations (e.g., eventual consistency).

*Principal corresponding author (1-519-888-4567 x31773).
**Corresponding author (1-519-888-4567 x32029).
Email addresses: h27fan@uwaterloo.ca (H. Fan), wgolab@uwaterloo.ca (W. Golab)

[1] Author supported by the Google Faculty Research Awards Program and the Natural Sciences and Engineering Research Council (NSERC) of Canada, Discovery Grants Program.

In this paper we focus on "black-box" testing of the consistency of a distributed storage system. The input is a history of operations applied to the system, and the output is a number that quantifies how far the history deviates from Herlihy and Wing's *linearizability* property [1]. Violations of linearizability, such as stale reads that fail to return the last updated value, can be quantified in units of time using the recently proposed Γ (Gamma) metric [2]. Such tests are more broadly applicable than rigorous proofs of correctness and model checking techniques, both of which assume knowledge of the system's internals and apply to an abstract model that may differ from the practical implementation.

Our contribution is an efficient algorithm for computing Γ given a history of read, write, and read-modify-write (e.g., conditional write) operations. The algorithm has time complexity $O(n^2)$ for a history of n operations, and generalizes a solution of Golab et al. for histories of reads and writes [2]. The simpler problem of deciding linearizability, which is equivalent to deciding whether Γ equals zero, was solved earlier by Misra [3] and by Gibbons and Korach [4]. Known algorithms for computing Γ and deciding linearizability efficiently assume a *read mapping*: for every operation that reads a value, the operation that wrote this value can be identified uniquely. Deciding linearizability is NP-complete otherwise [4], and solvable using state space exploration [5].

2. Preliminaries

Similarly to [1, 4], we formalize the observed behavior of a storage system as a *history*—a sequence of *events* representing invocations and responses of *operations*. *Linearizability* is a widely-adopted correctness condition for histories [1]. Informally, it states that one can assign for each operation a distinct *linearization point (LP)* between its invocation and response where it "appears to take effect." More precisely, if operations on a given object are ordered according to their LPs, their responses must be consistent with some sequence of state transitions permitted by the object's sequential specification. For example, updates appear to take effect serially, and reads always return the last updated value.

To quantify linearizability violations in units of time, we generalize the definition of a history from [1] by assuming that events are ordered by explicit timestamps. The invocation and response timestamp of an operation *op* are denoted as $inv(op)$ and $rsp(op)$, respectively. The *time interval* for such an *op* is the closed interval $[inv(op), rsp(op)]$. The Γ-*relaxation* of a history is obtained by shifting the invocation and response times by $-\Gamma/2$ and $+\Gamma/2$ time units, respectively. The Γ-*value* of a history is the minimum Γ for which the Γ-relaxation of the history is linearizable [2].

Because linearizability is a *local* property [1], the Γ-relaxation of a history is linearizable if and only if, for each object x, the subhistory of the Γ-relaxation comprising operations applied to x is linearizable. Therefore, to compute the Γ-value of a history H it suffices to compute the Γ-value of each subhistory where all operations are applied to a common object, and then obtain the maximum. The subhistory that requires the greatest Γ-relaxation determines the Γ-value of H.

In this paper, we consider three types of operations on objects: a *read* of value v is denoted (R, v); a write of value v is denoted (W, v); and a *read-modify-write* (RMW) that atomically reads v_r and then writes v_w, $v_w \neq v_r$, is denoted (RW, v_r, v_w).[2] An operation (W, v_w) or (RW, v_r, v_w) is called *dictating* with respect to an operation (R, v'_r) or (RW, v'_r, v'_w) if $v_w = v'_r$.

We make several assumptions regarding histories: (A1) following [2, 4], each operation has both an invocation and a response event; (A2) the history begins with a write operation that assigns the initial value of the shared object, and that does not overlap in time with any other operation; (A3) each write or RMW operation assigns a unique value; (A4) each read or RMW operation has exactly one dictating operation; and (A5) two RMW operations never read the same value. Assumption A4 establishes the read mapping and circumvent NP-completeness. Assumptions A1, A4 and A5 ensure that the history can be made linearizable by way of Γ-relaxation alone, as opposed to by adding or removing

[2] An unsuccessful Compare-And-Swap operation is represented by a read.

operations, and this implies that the Γ value of the history is well-defined.

To express our result, we borrow a number of definitions from [4]. For any value v, the *cluster for* v, denoted C_v, is the set of all operations that read value v, as well as their unique dictating operation.[3] The *zone* for a cluster C_v, denoted Z_v, is the closed interval of time between $Z_v.minrsp = \min_{op \in C_v} rsp(op)$ and $Z_v.maxinv = \max_{op \in C_v} inv(op)$. Intuitively, Z_v is a minimal subset of points in time where the LPs of operations in C_v may be chosen. A zone Z_v is called *forward* if $Z_v.minrsp < Z_v.maxinv$, meaning that the object had value v *continuously* from time $Z_v.minrsp$ to time $Z_v.maxinv$. Z_v is called *backward* if $Z_v.minrsp \geq Z_v.maxinv$, meaning that all operations in C_v overlap over the zone, and so the object had value v at least *at some point* inside the zone.

Gibbons and Korach characterize linearizability for histories of reads and writes as the absence of *conflicts* among pairs of zones [4]. Conflicts occur when two forward zones overlap, or when a backward zone is contained entirely within a forward zone. For histories that contain RMW operations, the clusters are first arranged into *cluster sequences* of the form $S = C_{v_1} C_{v_2} \ldots C_{v_k}$ where $(W, v_1) \in C_{v_1}$ and $(RW, v_{i-1}, v_i) \in C_{v_i}$ for $1 < i \leq k$.[4] A zone is defined for each cluster sequence S as the interval between $S_{minrsp} = \min_{op \in C_i, C_i \in S} rsp(op)$ and $S_{maxinv} = \max_{op \in C_i, C_i \in S} inv(op)$. The history is linearizable if and only if: (i) there are no conflicts among pairs of zones representing cluster sequences; and (ii) each cluster sequence is linearizable. The interval structure makes it possible to decide linearizability in $O(n \log n)$ time for a history of n operations [4].

3. Results

In this section we present our novel algorithm for computing Γ in histories of read, write and RMW operations. The main technical challenge lies in characterizing linearizability solely in terms of conflicts among zones; this reduces the problem of computing Γ to deciding the minimum Γ-relaxation required to

[3] An RMW operation is always part of two clusters under assumption A4.

[4] Assumption A2 ensures that (W, v_1) exists to form the cluster sequence.

remove every conflict [2]. Because Gibbons and Korach's verification algorithm (discussed at the end of Section 2) is only partially zone-based, we first modify it by introducing a new conflict type—*descendant-precedence*—to model conflicts within a single cluster sequence. Following [2], we also consider the conflict of a zone with itself when a value is read before it is written. The final Γ algorithm based upon these ideas appears in Section 3.2.

3.1. Linearizability verification using conflict detection

We first characterize linearizability for reads, writes and RMW operations in terms of zone conflicts alone. We say that a zone Z_v *precedes* a zone $Z_{v'}, (v \neq v')$, denoted by $Z_v \prec Z_{v'}$, if and only if $Z_v.maxinv < Z_{v'}.minrsp$. The negation of $Z_v \prec Z_{v'}$ (i.e., *does not precede*) is denoted by $Z_v \not\prec Z_{v'}$. Intuitively, $Z_v \prec Z_{v'}$ means that for some choice of LPs, all the operations in C_v take effect before any of the operations in $C_{v'}$; while $Z_v \not\prec Z_{v'}$ implies that for every choice of LPs, some operation in C_v must take effect after some operation in $C_{v'}$.

We say that a zone $Z_{v'}$ *inherts from* zone Z_v, denoted $Z_v \rightarrow Z_{v'}$, if and only if the RMW operation (RW, v, v') exists in the history. In that case, Z_v and $Z_{v'}$ are called *parent zone* and *child zone*, respectively. Inheritance is a transitive property. We use $Z_v \rightarrow^+ Z_{v'}$ (transitive closure of $\rightarrow$) to denote that Z_v is an *ancestor* of $Z_{v'}$, and $Z_{v'}$ is a *descendant* of Z_v, meaning that either (1) $Z_v \rightarrow Z_{v'}$; or (2) $\exists Z_{v''}$ *s.t.* $Z_v \rightarrow^+ Z_{v''}$ *and* $Z_{v''} \rightarrow Z_{v'}$.

Lemma 1. *For any linearizable history H, if $Z_v \rightarrow^+ Z_{v'}$, then $Z_v \prec Z_{v'}$.*

Proof. Without loss generality, we represent $Z_v \rightarrow^+ Z_{v'}$ as $Z_{v_1} \rightarrow \ldots \rightarrow Z_{v_n}$ $(v_1 = v,\ v_n = v')$. We prove $Z_{v_1} \prec Z_{v_n}$ by induction on n. (1) Basis: in a linearizable history, the RMW operation (RW, a, b) must take effect before every other operation in C_b, and after every other operation in C_a. Hence, except for the common RMW operation, all operations in the parent cluster take effect before all operations in the child cluster. If $n = 2$, then $Z_{v_1} \prec Z_{v_2}$ because Z_{v_1} is the parent zone of Z_{v_2}. (2) Assume that $Z_{v_1} \prec Z_{v_{k-1}}$ $(2 < k \leq n)$, then all operations in C_1 must take effect before (RW, v_{k-2}, v_{k-1}). Based on

$Z_{v_{k-1}} \rightarrow Z_{v_k}$, all operations in C_k must take effect after (RW, v_{k-2}, v_{k-1}). Combining these two orders, we can conclude $Z_{v_1} \prec Z_{v_k}$ (i.e., $Z_v \prec Z_{v'}$). □

We define a *descendant-precedence conflict* if two zones representing clusters satisfy $Z_v \rightarrow^+ Z_{v'}$, but $Z_v \not\prec Z_{v'}$. By Lemma 1, any history H containing such a conflict is not linearizable. To facilitate presentation, we label all the conflicts under consideration as follows, with examples shown in Figure 1:

C1. The above descendant-precedence conflict ($Z_v \rightarrow^+ Z_{v'}$ but $Z_v \not\prec Z_{v'}$).

C2. The intra-cluster conflict defined in [2] (in a given cluster C_v, $Z_v.minrsp$ is less than the start time of the dictating operation for C_v).

C3. The inter-cluster-sequence conflict defined in [4] for zones representing cluster sequences (two forward zones overlap, or a backward zone is enclosed entirely in a forward zone).[5]

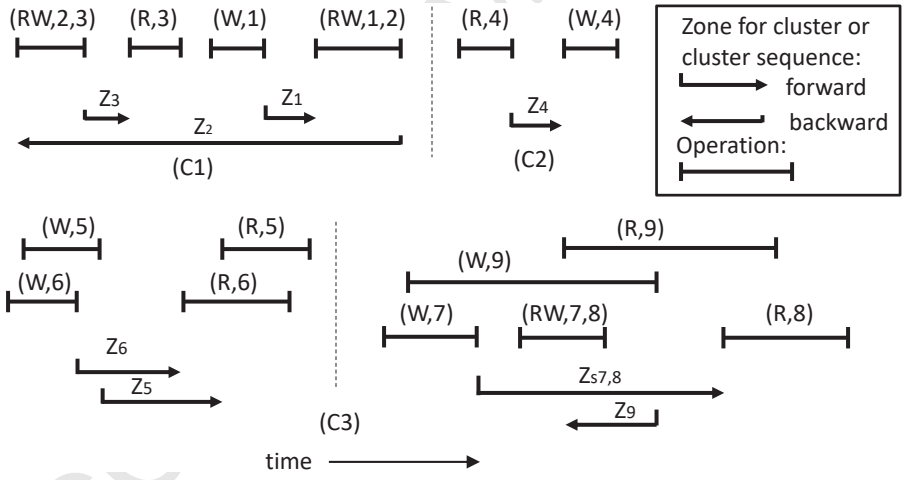


Figure 1: Examples of the three categories of conflicts.

Lemma 2. *Let H be a history composed of operations of a cluster sequence $S = C_{v_1} C_{v_2} \ldots C_{v_k}$. If H is free of conflicts C1 and C2, then H is linearizable.*

[5] A cluster without any RMW operations has a cluster sequence of length one.

PROOF. S can be represented as $Z_{v_1} \rightarrow Z_{v_2} \rightarrow \ldots Z_{v_k}$. Then $\forall i, j \in [1, k]$ $(i < j)$, $Z_{v_i} \rightarrow^{+} Z_{v_j}$ by definition. Because there is no conflict C1, $Z_{v_i} \prec Z_{v_j}$ holds. Then $Z_{v_i}.maxinv < Z_{v_j}.minrsp$ by the definition of $\prec$, and so there is no conflict among Z_{v_i} and Z_{v_j}. History H can be linearized as follows: in a forward zone Z_v, the dictating operation takes effect at $Z_v.minrsp$, and the remaining operations (except the RMW that is shared with the next cluster in S) take effect within Z_v; in a backward zone Z_v, operations take effect after any operations of ancestor zones and before any operations of descendant zones, at points where the backward zone does not overlap with any forward zone. □

Theorem 3. *If a history H is free of conflicts C1, C2, and C3, then H is linearizable.*

PROOF. Absence of C1 and C2 implies that operations within each cluster sequence are linearizable by Lemma 2, and may take effect in the order described in the proof of Lemma 2. As there is no conflict C3 for any pair of cluster sequences, the operations of a cluster sequence S can be linearized throughout the interval $[S.minrsp, S.maxinv]$ if S has a forward zone, and around one point in the interval $[S.maxinv, S.minrsp]$ where the backward zone does not overlap with any forward zone if S has a backward zone. Thus, H is linearizable. □

3.2. Γ computation for read, write, RMW operations history

Because RMW operations introduce the new challenge of dealing with conflict C1, the Γ computation algorithm in [2] does not work "out of the box" for a history of read, write, and RMW operations. For example, if we apply this algorithm to the example C1 in Figure 1, the result is a Γ-value of 0 as there is no conflict C2 or C3. However, the example is not linearizable. We propose a novel Γ computation algorithm, generalizing the previous technique [2] to accommodate RMW operations. The main idea is testing each type of conflict in the history and deciding on the minimum Γ necessary to resolve that conflict by way of Γ-relaxation. The pseudo-code is presented in Algorithm 1.

Algorithm 1: Γ computation for read, write, and RMW operations.

Input: History H that satisfies assumptions A1, A2, A3, A4 and A5

Output: Γ value of H

1 Process the history H into clusters $C_{v_1}, \ldots, C_{v_m}$ and identify the zone for each cluster as Z_{v_i} $(1 \leq i \leq m)$.

2 If any zone Z_{v_i} from step 1 has conflict C2, define the Γ score for v_i as $\Gamma_{C2}(v_i, H) = (inv(\text{dictating op. of } C_{v_i}) - Z_{v_i}.minrsp)$ to overcome the conflict; otherwise $\Gamma_{C2}(v_i, H) = 0$. Let $\Gamma_{C2}(H) = \max_{i \in [1,m]} \Gamma_{C2}(v_i, H)$.

3 Organize the clusters from step 1 into cluster sequences $S_1, \ldots, S_k$ and identify the zone for each cluster sequence.

4 For any cluster sequence S_i, if any pair of clusters $C_x, C_y \in S_i, (Z_x \rightarrow^+ Z_y)$ has a conflict C1, define the Γ score $\Gamma_{C1}(x, y, H) = (Z_x.maxinv - Z_y.minrsp)$ to overcome the conflict, otherwise let $\Gamma_{C1}(x, y, H) = 0$. Define the Γ score for cluster sequence S_i as $\Gamma_{C1}(i, H) = \max_{C_x, C_y \in S_i} \Gamma_{C1}(x, y, H)$.
Let $\Gamma_{C1}(H) = \max_{i \in [1,k]} \Gamma_{C1}(i, H)$.

5 If any pair of zones Z_i, Z_j for cluster sequences S_i, S_j identified in step 3 has a conflict C3, define the Γ score for the pair S_i, S_j as $\Gamma_{C3}(i, j, H) = \min(S_i.maxinv - S_j.minrsp, S_j.maxinv - S_i.minrsp)$ to overcome the conflict, otherwise let $\Gamma_{C3}(i, j, H) = 0$.
Let $\Gamma_{C3}(H) = \max_{i,j \in [1,k](i \neq j)} \Gamma_{C3}(i, j, H)$.

6 Return $\Gamma(H) = \max(\Gamma_{C1}(H), \Gamma_{C2}(H), \Gamma_{C3}(H))$.

Theorem 4. *Algorithm 1 returns the* Γ *value of history* H *in* $O(n^2)$ *running time, where* n *denotes the number of operations in* H.

Proof. Assume the algorithm outputs value γ for history H. Based on steps 2, 4 and 5 of the algorithm, there is no conflict C1, C2, C3 in the history H', which is the γ-relaxation of H. By Theorem 3, H' is linearizable.

To prove that γ is optimal, suppose for contradiction that there exists a $\gamma' < \gamma$ such that the γ'-relaxation of H is linearizable. Since Algorithm 1

returns γ, step 2, 4 or 5 of the algorithm detects a conflict in H that requires a γ-relaxation to resolve. A γ'-relaxation cannot resolve this conflict, contradicting the assumption that Algorithm 1 outputs the Γ-value of H.

Running time: Step 1 takes $O(n \log n)$ running time as in [2]. As the maximum number of zones is n, and comparing $Z.minrsp$ and dictating operation takes $O(1)$ time, step 2 takes $O(n)$ time. Step 3 can chain clusters into cluster sequences in $O(n^2)$ time by repeatedly checking for parent-child cluster pairs. The zone pairs check in step 4 and step 5 takes $O(n^2)$ time, and the step 6 takes $O(1)$ time. Thus, the total running time of the algorithm is $O(n^2)$, which is the same as the algorithm for histories of reads and writes [2]. □

4. Discussion and Conclusions

To our knowledge, this is the first paper that considers efficient computation of the Γ metric for histories containing RMW operations in addition to reads and writes. RMW operations such as conditional updates are an emerging feature in storage systems that support a mixture of weak and strong consistency models, and are currently supported in Apache Cassandra 3.0 and Amazon's SimpleDB.

References

[1] M. P. Herlihy, J. M. Wing, Linearizability: A correctness condition for concurrent objects, ACM Trans. Program. Lang. Syst. 12 (3) (1990) 463–492.

[2] W. Golab, M. R. Rahman, A. AuYoung, K. Keeton, I. Gupta, Client-centric benchmarking of eventual consistency for cloud storage systems, in: Proc. of the 34th ICDCS, 2014, pp. 493–502.

[3] J. Misra, Axioms for memory access in asynchronous hardware systems, ACM Trans. Program. Lang. Syst. 8 (1) (1986) 142–153.

[4] P. B. Gibbons, E. Korach, Testing shared memories, SIAM J. Comput. 26 (4) (1997) 1208–1244.

[5] K. Kingsbury, Knossos, `https://github.com/aphyr/knossos`, accessed: 2017-11-20 (2017).