THE PRESS RELEASE

"Three features that distinguish Chord from many peer-to-peer lookup protocols are its simplicity, provable correctness, and provable performance."

THE (NEWLY DISCOVERED) REALITY

- the only "proof" covers the join-and-stabilize case only, with no failures
- this "proof" is an informal construction of ill-defined terms, unstated assumptions, and unjustified or incomprehensible steps

  however, the subset can be proven correct, formally

- the full protocol is incorrect, even after bugs with straightforward fixes are eliminated
- not one of the six properties claimed invariant for the full protocol is invariantly true
- some of the many papers analyzing Chord performance are based on false assumptions about how the protocol works

USE

LIGHTWEIGHT MODELING
and avoid embarrassment!
THE FAIL EVENT

BEFORE

9

16

22

35

succ2

failing

AFTER

9

16

35

THE RECONCILIATION OPERATION

BEFORE

9

16

35

update: replace dead successor by live succ2

flush: remove dead predecessor

AFTER

9

16

35

reconcile: improve succ2 by replacing with successor's successor
ANTECEDENT PREDECESSORS

pred AntecedentPredecessors [t: Time] {
  all n: Node | let antes = (succ.t).n |
  n.prdc.t in antes
}

WHERE DID IT COME FROM?

must be an invariant to prove that the pure-join model is correct

WAS IT PREVIOUSLY KNOWN?

no, supporting my allegation that the previous "proof" is useless

IS IT GOOD FOR ANYTHING ELSE?

yes, it enables us to diagnose and fix a Chord bug
PROPERTIES CLAIMED INVARIANT
FOR THE FULL MODEL

after untangling bad definitions and formalizing, we get 5 properties

OneOrderedCycle
ConnectedAppendages

NOT ONE of these properties is actually an invariant!

ValidSuccessorList
(OrderedAppendages)
(OrderedMerges)

(Will be explained)
ORDERED MERGES

pred OrderedMerges [t: Time] {  
let cycleMembers =  
{ n: Node | n in n.^((bestSucc.t)) } |  
all disj n1, n2, n3: Node |  
( n3 in n1.bestSucc.t  
  && n3 in n2.bestSucc.t  
  && n1 in cycleMembers  
  && n2 !in cycleMembers  
  && n3 in cycleMembers  
) => Between[n1,n2,n3]  
}

The good news:  
Violations are repaired by stabilization.

The bad news:  
Compromises some lookups.  
Invalidates some assumptions used in performance analysis.

easily violated, even in the pure-join model
ORDERED APPENDAGES

WHY A POWERFUL ASSERTION LANGUAGE IS NEEDED

pred OrderedAppendages [t: Time] {
    let members = { n: Node | Member[n,t] } |
    let cycleMembers = { n: members | n in n.(^(bestSucc.t)) } |
    let appendSucc = bestSucc.t - (cycleMembers -> Node) |
    all n: cycleMembers |
        all disj a1, a2, a3: (members - cycleMembers) + n |
            ( n in a1.(^appendSucc) |
                && a2 = a1.appendSucc |
                && (a1 in a3.(^appendSucc) || a3 in a2.(^appendSucc) ) |
            ) => ! Between[a1,a3,a2]
}
"if a node x's successors skip over a live node y, then y is not in the successor list of any x antecedent"

how it can be violated

why it matters

valid successor list

20 joins, 17 stabilizes, 9 reconciles

17 stabilizes

9 stabilizes

17 fails, 13 updates

20 was part of the cycle, is now an appendage
WHY THE FULL PROTOCOL IS NOT CORRECT

DESIRED THEOREM: In any reachable state, if there are no subsequent joins or failures, then eventually the network will become ideal and remain ideal.

this ring is ideal

3 nodes join and become integrated

this ring is disordered, so the protocol cannot fix it

new nodes fail, old nodes update

this is actually a class of counterexamples:

- any ring of odd size becomes disordered
- any ring of even size splits into two disconnected subnetworks (which the protocol cannot fix)
## COMPARISON, REVISITED

<table>
<thead>
<tr>
<th><strong>PROMELA/SPIN</strong></th>
<th><strong>ALLOY</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>state structure</strong></td>
<td><strong>primal in Promela; displayed poorly by Spin</strong></td>
</tr>
<tr>
<td><strong>invariants</strong></td>
<td><strong>except for the most basic ones, an invariant must be written as a C program</strong></td>
</tr>
<tr>
<td></td>
<td>sometimes searching for the right invariant requires a great deal of trial and error—this is why C programs don't make good invariants</td>
</tr>
</tbody>
</table>

---

at least two studies of Chord have been made using the model checker Mace, and they did not find any of these problems

- very few, very weak invariants, so Mace did not have much to look for
- working on Chord implementations, so Mace could only do heuristic checking, not complete checking