UNDERSTANDING AND IMPLEMENTING CHORD

Pamela Zave

Princeton University
Princeton, New Jersey
The Chord protocol maintains a peer-to-peer network.

Identifier of a node (assumed unique) is an m-bit hash of its IP address. Here, the identifier space consist of all 6-bit binary values.

Nodes are arranged in a ring, each node having a successor pointer to the next node (in integer order with wraparound at 0).

Redundant pointers support fault-tolerance (extra successors, predecessors).

“Consistent hashing” means that all IP addresses are spread evenly around the identifier space.

The protocol preserves the ring structure as nodes join, leave silently, or fail.
CHORD AS A DISTRIBUTED HASH TABLE

a hash table stores (key, value) pairs

the keys are the same as identifiers

keys 22 through 32 are stored at 32

keys 33 through 42 are stored at 42,

etc.

to look up a value, you only need to know the IP address of one Chord member

* a list of accessible members is the only central administration needed!

your query to that member is forwarded around the ring until it gets to the member that has the value

* as an optimization, “finger” pointers go across the ring, make lookup faster
an operation changes the state of one member

Join and Stabilize are scheduled autonomously, GetsNotified is caused by another member’s Stabilize

now 10 is completely incorporated into the ring!
MORE OPERATIONS OF THE PROTOCOL

22 fails or leaves

22 has no pointers, does not respond to queries, so other nodes know that it has failed.

16 updates

35 flushes

9 reconciles

now the hole left by 22 is repaired.
WHAT YOUR IMPLEMENTATION MIGHT DO

16 FAILS

the ring is broken, and the protocol cannot recover it

now 3 has no pointer to another Chord member
nodes should reconcile more often, so their redundant successors have good information.

before over-writing 20 with 16, 3 should have checked that the new node is live.
DO YOU THINK YOU WOULD FIND AND FIX
ALL THESE LITTLE PROBLEMS?

DHTS (there are others besides Chord) have a reputation for being unreliable

*when a distributed system fails, it is very difficult to find out why!*
Donald Kossmann writes about an experiment he did with M.S. students in computer science at ETH in 2008:

“As part of a lab project on distributed systems, I asked the 8 students to implement the Chord protocol based on the Chord paper [same one you read]. Each student had his/her own implementation. The ETH students are pretty good systems builders so they all did fairly well and got something running: We had a small test suite and all implementations passed that.

Then, I asked the students to do the second part of the project:

- Run a Chord ring with your own implementation X.
- Then add a new node to this ring, but that new node uses Chord implementation Y from another student.

There was no combination of X and Y (not even if there was only one X node and one Y node) in which two nodes with different Chord implementations could even exchange a message. The interesting thing was that the problem was not serialization or message format or use of different TCP libraries or so. Those would have been easy fixes. There was no way for the students to fix the problem and get anything running even though they would get extra credit if they succeeded.

We allocated 10 weeks for the first part and 4 weeks for the second part. 4 weeks were not enough!”
WHY IS CHORD IMPORTANT TO YOU?

the 2001 SIGCOMM paper introducing Chord is one of the most-referenced papers in computer science, . . .

. . . and won SIGCOMM’s 2011 Test of Time Award

APPLICATIONS

- allows millions of ad hoc peers to cooperate

- WIDELY used as a building block in distributed fault-tolerant applications

  if you are a Ph.D. student, your dissertation may use Chord!

- the best-known application is BitTorrent

WHY MIGHT YOU CHOOSE CHORD TO IMPLEMENT YOUR DISSERTATION IDEA?

“Three features that distinguish Chord from many other peer-to-peer lookup protocols are . . .

. . . its simplicity,

. . . provable correctness,

. . . and provable performance.”
WHY IS CHORD IMPORTANT TO YOU?

the 2001 SIGCOMM paper introducing Chord is one of the most-referenced papers in computer science, . . .

. . . and won SIGCOMM’s 2011 Test of Time Award

APPLICATIONS

- allows millions of *ad hoc* peers to cooperate
- WIDELY used as a building block in distributed fault-tolerant applications
  
  *if you are a Ph.D. student, your dissertation may use Chord!*
- the best-known application is BitTorrent

WHY MIGHT YOU CHOOSE CHORD TO IMPLEMENT YOUR DISSERTATION IDEA?

“Three features that distinguish Chord from many other peer-to-peer lookup protocols are . . .

- . . . its simplicity, **TRUE!**
- . . . provable correctness, **OOPS!**
- . . . and provable performance.” **TRUE!**
THE CLAIMS

Correctness Property:

In any execution state, IF there are no subsequent Join or Fail events, . . .

. . . THEN eventually . . .

. . . all pointers in the network will be globally correct, and remain so.

THE REALITY

- even with simple bugs fixed and optimistic assumptions about atomicity, the original protocol is not correct

- of the seven properties claimed invariant of the original version, not one is actually an invariant

not surprisingly, due to sloppy informal specification and proof

Chris Newcombe and others at AWS credit this work with overcoming their bias against formal methods, which they now use to find bugs.

[I found these problems by analyzing a small Alloy model]

[CACM, April 2015]
"EVENTUAL REACHABILITY" IS NOT THE ONLY ISSUE

OrderedMerges . . .

. . . means that appendages merge in the correct places, as they do here

OrderedMerges is easily violated

VIOLATIONS OF OrderedMerges

- are not incorrect
- invalidate some assumptions used in performance analysis
- can be demonstrated in Chord networks with 3 nodes

how could they go unknown for ten years?

this is why formal methods are so important
extended successor list (ESL) of 29 (with $L = 2$):

29 41 55

member itself  dead  best successor (first live successor)

Original operating assumption:

No failure leaves a member without a live successor.

But if an ESL with $L = 2$ is . . .

29 32 29

. . . then 32 cannot fail!

Clearly, with $L = 2$, Chord is intended to tolerate one failure in a neighborhood, but not two.
extended successor list (ESL) of 29 (with L = 2):

\[
\begin{array}{ccc}
29 & 41 & 55 \\
\end{array}
\]

Definition of **FullSuccessorLists**: The extended successor list of each member has \(L+1\) distinct entries.

New operating assumption:

If a Chord network has the property **FullSuccessorLists**, then no failure leaves a member without a live successor.
extended successor list (ESL) of 29 (with L = 2):

Definition of \textit{FullSuccessorLists}:
The extended successor list of each member has L+1 distinct entries.

New operating assumption:

If a Chord network has the property \textit{FullSuccessorLists}, then no failure leaves a member without a live successor.

if not satisfied for the real failure rate,

\ldots increase rate of stabilization,

\ldots or increase redundancy
TO MAKE ORIGINAL CHORD CORRECT:

- alter the initialization to satisfy *FullSuccessorLists* with all members live
  - requires L+1 members
- alter the operations to populate successor lists more eagerly, preserve *FullSuccessorLists* at all times

```
now it is roughly correct (in hindsight) but how do we prove it without an invariant?
```
WHY IS FINDING AN INVARIANT SO DIFFICULT?

THE KNOWN, NECESSARY PROPERTIES ARE STATED IN TERMS OF THE RING . . .

- there is a ring of best successors
- there is no more than one ring
- on the unique ring, the members are in identifier order
- from each appendage member, the ring is reachable through best successors

. . . BUT “RING VERSUS APPENDAGE” IS CONTEXT-DEPENDENT AND FLUID:
AN INTERMEDIATE RESULT

ANOTHER OPERATING ASSUMPTION:
A chord network has a *stable base* of L+1 nodes that are always members.

I believe it is just preventing anomalies in small networks, but how can we know for sure?

THE INDUCTIVE INVARIANT:

OneOrderedRing

and ConnectedAppendages

and BaseNotSkipped

THE PROOF OF CORRECTNESS:
by exhaustive enumeration, in Alloy, for all model instances up to N = 9, L = 3

expensive to implement these high-availability nodes!

a stable base would have 3-6 members, while a Chord network can have millions of members—what is the base doing?
THE FINAL RESULT

ANOTHER OPERATING ASSUMPTION:

None

THE INDUCTIVE INVARIANT:

OneLiveSuccessor and SufficientPrincipals

this is just a formalization of the original operating assumption

Definition of a principal member:
A member that is not skipped by any member’s successor list.

Definition of SufficientPrincipals:
There are at least L+1 principal nodes.

THE PROOF OF CORRECTNESS:

informal and intuitive, but . . .

. . . a real proof (no size limits)

. . . backed up by an Alloy model checked up to N = 9, L = 3
(as a protection against human error)
CONCLUSIONS

SPECIFICATION OF A CORRECT VERSION OF CHORD

● initialization is more difficult than original Chord, but a simple protocol will get networks off to a safe start

● otherwise correct Chord is just as efficient as original Chord

  it is an impressive pattern for fault-tolerance

● these peer-to-peer protocols have a (justified) reputation for unreliability

● a correct specification could pave the way for a new generation of reliable, more useful implementations

  it also provides a firm foundation for work on better failure detection and security

FOR YOUR DISSERTATION, ONLY THE BEST WILL DO!

use an implementation based on

“Reasoning about identifier spaces: How to make Chord correct”, Pamela Zave,
IEEE Transactions on Software Engineering
APPENDIX:

AN OUTLINE OF THE PROOF
Definition of OrderedSuccessorLists:
For all distinct identifiers \( x, y, z \),
and sublists \([x, y, z]\) of an ESL
(whether the sublist is contiguous or not) . . .

\[ \text{hypothesize a disordered extended successor list } \]
\[ [. . . x, . . . y, . . . z, . . .] \]

\[ \text{between } [x, y, z]. \]

Proof of OrderedSuccessorLists

\[ [x, . . . y, . . . z] \] must include \( L + 1 \) principal nodes

\[ \text{but the length of an ESL is always } L + 1 \]

\[ \text{CONTRADICTION!} \]
The principal nodes make the shape of the ring

Every member has a best successor (first live successor)

There are sufficient principal nodes

Here is a graph of best successors:

These paths . . .

. . . do not skip principal nodes

. . . are acyclic

. . . are ordered by identifiers

Each tree has exactly one \( p_s \), which is unique to it

So the re-arranged graph must look like this

Automatically satisfying \textit{OneOrderedRing} and \textit{Connected-Appendages}
AN OPERATION IS A SEQUENCE OF ATOMIC STEPS

EACH ATOMIC STEP IS AN INTERNAL STATE CHANGE OR THIS:

while waiting for a reply (or timeout), X cannot answer queries about its state

because of the structure of operations, queries cannot form circular waits
HOW STABILIZE PRESERVES THE INVARIANT

extended successor list of stabilizing node, before Stabilize

extended successor list of its new successor

because invariant holds, no current principal nodes are skipped here or here

precondition guarantees that between \([S_0, N_0, S_1]\), so no current principal nodes are skipped from \(S_0\) to \(N_0\)

therefore, no former principal nodes are skipped by this new successor list, and the number of principal nodes has not decreased

some Chord operations need multiple atomic steps

in the new, provably correct, specification, every intermediate operation state is also constructed in this safe way

JOIN AND RECTIFY ARE SIMILAR
HOW FAIL PRESERVES THE INVARIANT

PRESERVATION OF OneLiveSuccessor

The operating assumption is that no failure leaves a member with no live successor, . . . .

. . . so the invariant is assumed to be preserved.

PRESERVATION OF SufficientPrincipals

Lemma: The only operation that can cause a node to change from principal to non-principal is its own failure.

Why can’t failure of a principal node leave the network with fewer than L+1 principals?

The life history of a long-lived member:

1. Join
2. become principal because all neighbors know you
3. enjoy life as a principal node
4. Fail

Therefore the number of principal nodes is proportional to the number of nodes.

Once the network has grown (especially to millions of members!) it is overwhelmingly improbable that it will have fewer than 3-6 principal nodes.
IF THERE ARE NO MORE JOIN OR FAIL EVENTS . . .

. . . WHILE MEMBERS CONTINUE TO STABILIZE . . .

1. dead successors are removed, so that every member’s first successor is live

2. every member’s first successor and predecessor become globally correct

3. tails of all successor lists become correct

as with construction of intermediate successor lists, operations must be specified precisely to ensure correctness

here preconditions must ensure that no operation reverses the progress of a past or current phase
CONCLUSIONS

THE PRODUCT

- Initialization is more difficult than original Chord, but a simple protocol will get networks off to a safe start.

- Otherwise correct Chord is just as efficient as original Chord.

- It is an impressive pattern for fault-tolerance.

- These peer-to-peer protocols have a (justified) reputation for unreliability.

- A correct specification could pave the way for a new generation of reliable, more useful implementations.

- It also provides a firm foundation for work on better failure detection and security.

THE PROCESS

- Chord is a very interesting protocol—note that the invariant looks nothing like the properties we care about!

- Results would have been impossible to find without model-checking to explore bizarre cases and get ideas from them.

- That is where the idea of a stable base came from.

- The best result was impossible to find without the insights that came from the proof process.

- Pamelazave.com > How to Make Chord Correct