Asynchronous Systems (CSE 535), fall 2013 **December 6, 2013**
Student ID: 109294122 **Stony Brook University**

Project Report for

Verification of properties of Chord: Alloy to TLA

Prasidh Srikanth

psrikanth@cs.stonybrook.edu

Contents

Chapter 1

Problem statement and plan

Introduction

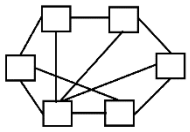

Figure 1: Peer to peer system

P**eer to Peer** systems are distributed systems without any centralized control in which individual nodes (say Computers) can act as both the client and the server for other nodes in the network. They allow shared access to several resources such as files, streaming audio and video without any need of a central server. A simple peer to peer network is shown in Figure 1 where the rectangles show the nodes and the connected nodes joined by lines. Peer to peer networks implement some form of virtual overlay network on top of the physical network topology for routing purposes. Based on how the nodes are linked to each other within the overlay network, they are classified as structured and unstructured. In order to efficiently lookup resources, a P2P system must be structured in a way that provides certain guarantees of correctness and performance. More commonly such structured P2P systems implement a **distributed hash table** to provide these guarantees, even if the resource is extremely rare [1].

A D**istributed Hash Table** is a class of decentralized distributed systems that provide a lookup service similar to a hash table; (Key, Value) pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key [2].

Chord is a protocol and algorithm for a peer to peer distributed hash table specifying how keys are assigned to nodes, and how a node can discover the value for a given key by first locating the node responsible for that key [3]. Chord is completely decentralized and symmetric, and can find data using only log (N) messages, where N is the number of nodes in the system. Chord's lookup mechanism is provably robust in the face of frequent node failures and re-joins [4].

Applications of chord DHT:

Systems that make use of chord DHT include CFS [6], UsenetDHT [7] and Overcite [8]. Cooperative File System allows anyone to publish and update their own file system, and provides read-only access to others; UsenetDHT allows Usenet servers to share storage instead of fully replicating articles locally; OverCite is a distributed version of the CiteSeer digital library. These

systems all take advantage of Chord to distribute the load of serving data very widely to achieve high performance despite encountering flash crowds.

Part 1: Problem Description

Modern operating systems make extensive use of distributed algorithms which are very subtle and easy to get wrong. Hence model checking is made use of to prove algorithm correctness which in turn helps to get a better understanding of the algorithm. I am planning to carry out formal verification of properties of chord using TLA by mimicking the modelling done by Pamela Zave in alloy [16]. TLA+ has been used to specify and check the correctness for several hardware level protocols. I would be using it to verify some properties of Chord and analyze them using TLA.

How I approached it: I initially understood TLA+ and gained an understanding of model checking. I then went through Pamela Zave's modeling of Chord in Alloy to get an understanding about how the invariants of chord were modelled [10]. With this understanding I have implemented Chord DHT on TLA+ and written few invariants of Chord.

Input: Code for chord in Pluscal translated into TLA+ for every invariant.

Output: Results display the spec status as parsed and the property being analyzed is correct.

Part 2: The Protocol

The distributed hash table Chord was first presented in a 2001 SIGCOMM paper [9]. This paper claimed that three features that distinguish Chord from many other peer-to-peer lookup protocols are its simplicity, provable correctness and provable performance. Despite these assurances, Pamela Zave has found inconsistencies and ambiguities in the claimed invariants and has come up with a correct version of the Chord protocol. The correct version of the protocol as proposed by Pamela Zave can eventually repair all disruptions in the ring structure, given ample time and no further disruptions while it is working.

She has made use of lightweight modeling to verify the correctness of the modified chord protocol. Lightweight modelling is the process of building a small, abstract formal model of the key concepts of the software system and analyze the model with a fully automated tool that works by exhaustive enumeration over a bounded domain of possibilities [10]. I am studying the model

written in Alloy for the correct version of Chord and analyzing it using the alloy analyzer as mentioned in [14].

In a Chord ring every member has an identifier that is an m-bit hash of its IP address. In addition to this, each member has a successor pointer, predecessor pointer and a pointer to its second successor. Each node in the chord ring knows at least the *id* of its immediate successor which in turn helps the nodes to look up other nodes in the network using this successor pointer. To speed up the look up, each node maintains its own finger table containing the address of up to m successors. Every time an event occurs, the ring maintenance protocol allows one member of the chord network to query other members and update the local pointer if there are better values.

New nodes are added to the chord ring whenever a **join** event occurs. The newly joining members contact an already existing member to get their successor id thereby forming appendages. Whenever appendages occur, the nodes have to **stabilize** and when a node in the chord ring stabilizes they do the following:

- Update the value of its successor's predecessor
- If the predecessor is closer in identifier order than its current successor, then update the predecessor as its new successor.

Stabilization essentially aims at incorporating new nodes into the Chord ring. Once stabilization is complete, a node ***notifies*** its successor of its presence which in turn updates the intimating node as its predecessor (provided its id is smaller than its current predecessor). Stabilize and notify events help in repairing any disruptions that may have occurred to the chord ring due to a join event.

Every node in the cord ring responds to queries as long it is alive. When a node ***fails***, it leaves the network silently and stops responding to any further requests. This helps in detecting nodes that have failed. A node in the chord ring would never fail if its failure would leave another node without any successor in its successor list.

Failures end up producing holes in the chord ring. This is taken care of with the help of events such as reconcile, update and flush which are periodically executed by each member. During a ***reconcile*** event, a node adopts its successor's successor as its second successor. This event is essentially that part of the protocol that helps recover from failure. When an ***update*** event

occurs, the node updating would replace its successor pointer to a dead member by the first successor pointer which is pointing to a live member. During a ***flush*** event, nodes discard their dead predecessor [10]. This is all about how the chord protocol functions.

Invariants in Chord:

There are seven invariants that are defined in the chord protocol and each of them uniquely play a role in the efficiency and correctness of the protocol. If there is a path connecting two nodes using the successor lists and finger tables, then the nodes are said to be connected. The spirit of connectivity can be expressed using the three invariants: **AtLeastOneRing** (the chord network has atleast one cycle), **AtMostOneRing** (the chord network is not broken into many cycles and comprises of a single ring) and **ConnectedAppendages** (incoming appendages are connected to the chord ring). To verify that the order imposed by successors is consistent with identifier order in a cycle we check for the **OrderedRing** invariant. To verify that the order imposed by successors is consistent with identifier order in an appendage we check for the **OrderedAppendage** invariant. The **OrderedMerges** invariant is used to check if an appendage merges into the ring at the right place in identifier order. **ValidSuccessorList** claims that if *v* and *w* are members, and if w's successor list skips over *v*, then *v* is not in the successor list of any immediate antecedent of *w*.

The invariants that are desirable for ensuring *key and data consistency* are OrderedMerges, OrderedAppendages and ValidSuccessorList. To ensure *correctness* of the Chord ring structure the invariants that are to be verified are ConnectedAppendage, AtLeastOneRing, OrderedRing and AtMostOneRing.

The initial version of Chord handled failure using reconcile, update and flush events. By making use of these events ambiguities were overcome only in the case of ConnectedAppendages but not for other invariants. Hence, Pamela Zave has come up with a better Chord implementation in which *stabilize* and *update* events include the effects of the reconcile events. In addition to this the specification of join and stabilize have also been modified providing better functionality [10].

Part 3: Best modeling of chord in alloy

The alloy language is powerful when used for expressing structural network properties. It has a combination of first-order predicate logic and relational algebra, with transitive closure built-in. Hence, it is made use of by Pamela Zave to verify the invariants of the chord protocol. The

main purpose of choosing alloy to describe the protocol is because it is easier to understand the properties, the Chord protocol seeks to restore [11]. Modelling in alloy has helped to avoid several ambiguities which arise during node joins and node failures because we can include preconditions in alloy which need to be satisfied for an event to occur.

The model in alloy is built by abstracting many implementation details while preserving central concepts of Chord like allowing nodes to read the state of other nodes. The following is the code fragment from the alloy modeling of correct version of Chord:

```
sig Node {
        succ: Node lone -> Time,
        succ2: Node lone -> Time,
        prdc: Node lone -> Time,
        bestSucc: Node lone -> Time
        }
```

The above fragment of alloy code says that there are individuals of type *Node.* At any given point of time, each of the nodes have zero or one pointers to succ (successor), succ2 (second successor), prdc (predecessor) and bestSucc (best successor).

```
pred OneOrderedRing [t: Time] {
  let ringMembers = { n: Node | n in n.^(bestSucc.t) } |
    Network.base in ringMembers      -- at least one ring containing base
  && (all disj n1, n2: ringMembers | n1 in n2.^(bestSucc.t) )   -- at most one ring
  && (all disj n1, n2, n3: ringMembers | n2 = n1.bestSucc.t => ! Between[n1,n3,n2]
                                                    -- ordered ring )
}
```

The above fragment of code is written to formalize the properties AtLeastOneRing, AtMostOneRing and OrderedRing. The set ringMembers contains all nodes that can reach themselves by following bestSucc pointer. The set must also be not empty as it would lead to no ring existing. Any two nodes of the Chord ring must be able to reach each other, otherwise it means

there is more than one ring in the network. If n1 and n2 are two nodes in the chord network and n2 is the best successor of n1, then no other node must be between them in identifier order [10].

Alloy analyzer has very good visualization tools which display instances (Figure 2 and Figure 3) and counterexamples as graphs. If an assertion is valid, then the analyzer returns an example of it (Figure4 and Figure5). If a property is not satisfied by a model, then the analyzer returns a counterexample that violates it.

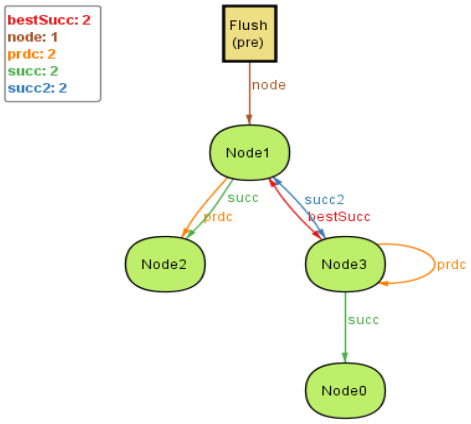

Figure 2: IsEffectiveFlush for 4 but 1 Event, 2 Time 0; Instance found. Predicate is consistent. 57ms.

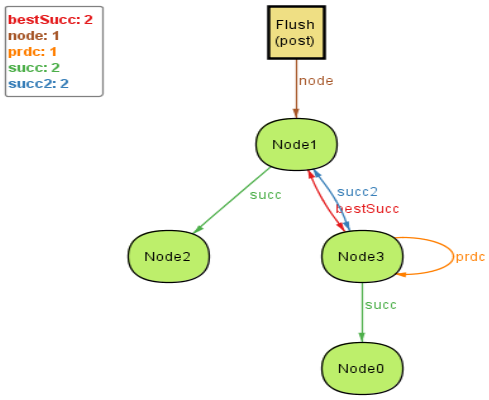

Figure 3: IsEffectiveFlush for 4 but 1 Event, 2 Time 1; *Instance found. Predicate is consistent. 57ms.*

Executing "Run IsEffectiveStabilize for 4 but 3 Event, 4 Time"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
28723 vars. 368 primary vars. 41807 clauses. 760ms.
Instance found. Predicate is consistent. 176ms.

Executing "Run IsEffectiveNotified for 4 but 3 Event, 4 Time"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
28723 vars. 368 primary vars. 41807 clauses. 611ms.
Instance found. Predicate is consistent. 173ms.

Figure 4: Alloy analyzer showing the link to an instance found for a predicate in full_chord_ routing in alloy

Executing "Check IdealImpliesSucc2Correct for 6 but 0 Event, 1 Time"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
4906 vars. 187 primary vars. 11541 clauses. 103ms.
No counterexample found. Assertion may be valid. 1103ms.

Executing "Check ValidImpliesHasBestSuccessors for 6 but 0 Event, 1 Time"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
4489 vars. 193 primary vars. 9333 clauses. 791ms.
No counterexample found. Assertion may be valid. 286ms.

Figure 5: Alloy analyzer verifying the assertion and coming up with no counterexample, thus the assertion may be valid.

Part 4: Project Plan

In this project, properties of Chord protocol are studied by making use of the lightweight modelling carried out by Pamela Zave in alloy. I am writing a few invariants in PlusCal [13] and carrying out model checking in TLA. TLA [12] is a tool used to describe behaviors of concurrent systems.

Phase 1: (Week 1 and 2)

- Project Proposal: Since this project involved a team of students working on a particular topic of interest, the first week was spent identifying the possible tasks for each member. Once I identified my task, I came up with a proposal containing the problem description and my plan of action.
- During the second week, I studied the properties of Chord using Pamela Zave's alloy implementation. I made use of the alloy analyzer to closely understand the models of the chord protocol implemented in alloy.

Phase 2: (Week 3 and 4)

- Once I understood the modelling in alloy, I got started with TLA and setup the tools required for the implementation. The initial couple of weeks was spent in coming up with a design for the implementation.
- Then I wrote a specification of Chord in PlusCal by mimicking the specification in alloy.

Phase 3: (Week 5)

- Write invariants of chord and carry out model checking using the TLA tools. Verify properties of Chord using the TLA model checker and verify for correctness.

Phase 4: (Week 6)

- Starting from the smallest invariants, all of them were recorded and incorporated in the report. Finally, the presentation for the project was created.

Chapter 2

Design

Overview

At a high level, the design of the project can be visualized as consisting of a structure of a ring network with few simplifications to the original chord version:

1. The node itself, its IP address, and its hashed identifier are all combined into one.
2. The successor list of a node has size 2.
3. Fingers are not included in the model, as they have no effect on ring structure, and the early fingers and successors coincide [16].

The modelling of the correct version of chord in alloy contains a permanent base set of member of size s+1, where s is the length of the successor list. The candidate invariant properties that are implemented in alloy include “One Ordered Ring”, “Connected appendages”, “antecedent predecessors”, “ordered appendages”, “ordered merges”, “distinct successors”, “ordered successors”, “Valid Successor List”, “ReachableSuccessors2” [14]. They in turn verify the spirit of Ordering, Connectivity etc. These are the only properties that need to be verified for an undisturbed network to become ideal through stabilization and reconciliation.

Why alloy?

We can build an abstract model in comparison to a real implementation and focus more on the key concepts of the protocol or algorithm under consideration. It is easy and fun to do (Gives intuitive examples and counter examples for all instances found).

In alloy, a state consists of sets and relations over individuals. Facts have been expressed combining relational algebra, first-order predicate logic, transitive closure and object orientation. All events can be specified with preconditions and post conditions. Time is given explicitly with:

- Timestamp marking at which time a tuple exists
- Every event having a pre-time and a post-time

In an alloy model, assertions are expressed through facts. A trace is specified by restricting its sequence, say "a request is followed by a response". Pamela Zave mentions that the ideal properties to verify in the chord protocol would be to test the stability of the ring and successor correctness [5].

Invariants for proving correctness:

An invariant is usually an incomplete description focusing only on the properties that are important for the system operation and needed for verification. The invariant of a routing protocol show how badly corrupted or out of date the routing information can become [21]. The invariants that I plan to verify are Ordered Ring, Ordered Appendages and Ordered Merges and Valid Successor List.

The specification of the correct version of chord in alloy centers on predicates "valid" and "ideal" expressing the valid and ideal states of the chord network. In this specification the **safety proof** establishes that the chord network never reaches an invalid state. The **Liveness proof** establishes that, starting from a valid state, if there is no further disruption, then the protocol will eventually bring the network to an ideal state [21].

What the invariants claim?

The invariant **"Ordered merges"** says that an appendage merges into the ring at the right place in identifier order. **"Ordered appendages"** conveys the message that members are ordered correctly within an appendage. The invariant **"Valid Successor List"** claims that if 'v' and 'w' are members, and if w's successor list skips over 'v', then 'v' is not in the successor list of any immediate antecedent of 'w'. The invariant **"Ordered Ring"** says that the ring must always be ordered by the identifiers. A state that satisfies these invariants is said to be valid.

Understanding and using TLA

A specification is a written description of what a system is supposed to do and helps to get a better understanding of the system. TLA helps to describe a system by providing a nice way to formalize the reasoning of systems using assertions [17].

I installed the TLA+ toolbox from [19] and analyzed a few sample programs to get an understanding of how TLA works. To get a better understanding of TLA+ I am referring to

Lamport's Specifying Systems [17]. TLA+ has been used to specify and check several hardware protocols and I am planning to use it to specify and check Chord protocol.

PlusCal is a language that is made use of to write formal specification of algorithms and is easy to describe the flow of an algorithm. We can make use of a TLA+ expressions in a PlusCal algorithm which makes it more expressive and helps in writing distributed algorithms. I will first write the algorithm in PlusCal and translate (compile) it into a TLA+ specification that can be checked with the TLA+ tools.

A typical TLA+ specification has the form *Init* ^ *Next* ^ *Liveness* where,

"Init" is the initial state predicate- a formula describing all legal initial states

"Next"is the next state relation which specifies all possible steps in the behavior of the system.

"Liveness" is a temporal formula that specifies the progress of the system as the conjunction of fairness conditions on actions [18].

This specification essentially describes a state machine.

I would be implementing my specification in TLA by mimicking existing specification of Chord in Alloy and making use of the invariants.

Chapter 3

Implementation

Overview

The project essentially aims at carrying out model checking of the Chord protocol in TLA. As mentioned in the design section, the first goal of the project is to analyze the modelling of the chord protocol in alloy by Pamela Zave. The next goal, pertaining to the implementation, in this project was to write the invariants in PlusCal and then translate it into TLA+ to get a good clarity of code. The implementation has been tested to work on Windows platform.

In the alloy model of chord, nodes correspond to identifiers and an alloy library component **ordering** is invoked to ensure that the members of a node are totally ordered. There are no finger tables and each chord event contains a node field whose value is the single node at which the event takes place. Join and stabilize events occur at any time whereas the notify event is preceded and caused by a stabilize event. Given a model, the alloy analyzer checks all possible instances. If all instances satisfy a given assertion, then the assertion is verified for that scope [21].

Liveness verification is carried out for a network of size s. The error of a successor or predecessor is defined as 0 if it points to the correct member in the network, 1 if it points to the next most correct member in the network, s-1 if it points to the least correct member, s if there is no pointer and s+1 if it points to a non-member. The error of the second successor is given as 0 if its nodes successor is live and it matches the successor of its nodes successor. If this is not satisfied the error is positive. The total error is given by the sum over all members [16].

In the implementation, the presence or absence of a successor determines whether a node is a member of the network or not. It makes the following simplification: the concept of node, its IP address and finger table are all combined such that nodes correspond to identifiers. The **between** predicate implemented is used to test the order of the node identifiers. The code has been documented using built-in TLA comments which serves as a good source of inline documentation.

Implementation Details

- **Platform:** Windows
- **Language:** Pluscal/TLA+
- **Tools:** TLA+ toolbox
- **Code size:** 258 lines of code

The alloy implementation of Chord is slightly larger than the original pseudo code because the alloy descriptions include extra information such as preconditions. This in turn helps us get a complete characterization of the possible effects on the network state by checking in the alloy analyzer [11].

For an ideal ring to recover from a single join two stabilize events and two notify events would be required. Similarly, for an ideal ring to recover from a single failure, one update event, two reconcile events, one flush event, one stabilize event and one notify event would be required to occur in the correct order. With this level of concurrency, the chord protocol has been shown to perform efficient by Pamela Zave in her alloy modelling of the protocol [21].

CREDITS and ACKNOWLEDGEMENT: The implementation was done by Jitender Kalra [22] and me together. We have expressed different properties of Chord in TLA.

The full source code is included in **[15]**.

Chapter 4

Testing and Evaluation

Overview

A new model is then created using the TLC model checker. This opens a model editor on the model and the editor has three pages. In the model overview page, the values for the declared constants are given and the invariants are added along with initial predicate and next state relation. The module is then parsed with the spec status showing "parsed" highlighting in green.

The model is run using TLC model checker which takes a few seconds and stops, reporting output details with the values printed on the console. This means that the specification is sensible determining a collection of behaviors. If the invariant does not hold, then the model checker throws an error message in the error window. This means that the invariant is wrong and we need to modify it.

Test setup and execution

Model Checking results:

Start time: Wed Dec 04 11:14:44 EST 2013

End time: Wed Dec 04 11:14:50 EST 2013

Current Status: Not running

Fingerprint collision probability: calculated: 1.6E-13, observed: 6.0E-14

Spec Status: Parsed

State space progress (click column header for graph)

Time	Diameter	States Found	Distinct States	Queue Size
2013-12-04 11:14:50	16	3842	1024	0
2013-12-04 11:14:48	8	1213	422	158

Figure 6: Statistics

In the statistics section of the TLC model checking results page, the state space progress table tells the number of distinct states TLC found. The **diameter** denotes the largest number of states the execution reached before it repeated a state, **states found** gives the total number of states examined in first step, **distinct state** gives the number of states that form the set of nodes and **queue size** specifies the total number of states(not necessarily distinct) that were examined.

Summary

Overall, the correctness of all the invariants of Chord modelled in alloy were thoroughly studied. All four versions of chord as specified by Pamela Zave were executed and analyzed. The instances generated for certain assertions when executing the full chord version in alloy were all found to be rectified in the modelling of the correct version of Chord. The key conclusions from the analysis in alloy is that the invariant properties are desirable for key consistency and to constrain the reachable states. With this understanding, specifications were written in TLA and verified.

From this project, I realized that a lightweight modelling tool like alloy is extremely efficient for modelling network structures like Chord and their invariants. It is much faster than implementation and testing for finding the design flaws.

Chapter 5

References:

[1] Wikipedia article on Peer-to-Peer (last modified 2 December 2013) http://en.wikipedia.org/wiki/Peer-to-peer

[2] Wikipedia article on Distributed Hash Table (last modified on 5 December 2013) http://en.wikipedia.org/wiki/Distributed_hash_table

[3] Wikipedia article on Chord (last modified on 1 October 2013) http://en.wikipedia.org/wiki/Chord_(peer-to-peer)

[4] Github repository of Chord development (last edited on March 2013) https://github.com/sit/dht/wiki

[5] Pamela Zave (April 2012), A correct version of Chord http://www2.research.att.com/~pamela/aCorrectChordTalk.pdf

[6] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica (2001), "Wide-area cooperative storage with CFS", http://pdos.csail.mit.edu/papers/cfs:sosp01/

[7] Emil Sit, Robert Morris and M.Frank Kaashoek (April 18, 2008). UsenetDHT:A low overhead design for Usenet https://www.usenix.org/legacy/event/nsdi08/tech/full_papers/sit/sit_html/

[8] Jeremy Stribling, Jinyang Li, Isaac G. Councill, M. Frans Kaashoek, and Robert Morris (2006), "A Distributed, Cooperative CiteSeer" http://pdos.csail.mit.edu/papers/overcite:nsdi06/index.html

[9] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan (August 2001). Chord: A scalable peer-to-peer lookup service for Internet applications. In Proceedings of SIGCOMM. ACM. http://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf

[10] Pamela Zave (April 2012), "Using Lightweight Modeling To Understand Chord", ACM SIGCOMM Computer Communication Review, http://www2.research.att.com/~pamela/chord-ccr.pdf

[11] Pamela Zave (October 2011), Experiences with Protocol Description http://www2.research.att.com/~pamela/wripe.pdf

[12] "The TLA home page" (15, June 2013) http://research.microsoft.com/en-us/um/people/lamport/tla/tla.html

[13] "The PlusCal algorithm language" (April, 21, 2011) http://research.microsoft.com/en-us/um/people/lamport/tla/pluscal.html

[14] On the correctness of chord, http://www2.research.att.com/~pamela/chord.html

[15] Implementation of Chord in TLA (December, 5, 2013), https://github.com/prasidh09/CSE-535---Chord-Distributed-Hash-Table

[16] Pamela Zave (April 2012), Correct version of chord in alloy, http://www2.research.att.com/~pamela/chordbestccr.als

[17] Leslie Lamport (June, 18, 2002), Specifying Systems, http://research.microsoft.com/en-us/um/people/lamport/tla/book-02-08-08.pdf

[18] Leslie Lamport, John Matthews, Mark Tuttle, Yuan Yu (September 2002), Specifying and verifying systems with TLA, http://research.microsoft.com/en-us/um/people/lamport/pubs/spec-and-verifying.pdf

[19] "The TLA+ Proof Systems" (latest version September 2013) http://tla.msr-inria.inria.fr/tlaps/content/Download/Binaries/Windows.html

[20] "The TLA+ Hyperbook" (Last Modified Nov 28, 2013) http://research.microsoft.com/en-us/um/people/lamport/tla/hyperbook.html

[21] Pamela Zave (January 2010), "Lightweight verification of network protocols: The case of Chord" http://www2.research.att.com/~pamela/chord-orig.pdf

[22] Jitendra Kalra (December 2013), "Chord-TLA" https://github.com/jitkalra8/Chord-TLA