University of Basel

Peer-to-Peer / Chord

Yiming Wu <yiming.wu@unibas.ch>

Databases and Information Systems Group, University of Basel

Exercise 7

University of Basel

Prof. Dr. Florina Ciorba
Prof. Dr. Heiko Schuldt
Prof. Dr. Christian Tschudin
Prof. Dr. Isabel Wagner

Foundations of Distributed Systems — Fall 2024

Exercise 7 — **Hand-in: 2024 December 8th (13:00pm)**

Advisors: Yiming Wu (yiming.wu@unibas.ch)

Modalities of work: The exercise is solved in teams of 2 people.

Modalities of the exercise: The solutions to the exercise must be uploaded to ADAM before the deadline and presented to the advisor. For the presentation, every student is required to schedule an appointment by reserving a slot on `https://xoyondo.com/dp/pzedfa6cfg99hd5`. Every slot has a capacity of two people and for groups, every group member must sign-up. Please note, that reservations are made on a "first come first served" basis. During the meeting, the work and the understanding of the technical background is evaluated. Therefore, each group member has to attend this meeting in order to be awarded points for the exercise. Please be prepared to demonstrate your solutions on your machine! If you need to have the interview via zoom, please send an email to Yiming Wu (yiming.wu@unibas.ch)

Introduction

In this exercise, you will implement a Peer-to-Peer (P2P) overlay network based on the CHORD algorithm. The network of nodes builds up a data store. The data should be distributed among all available nodes in the network. Each *data item* is represented as a *key-value pair*: the *key* uniquely identifies the data item and is used to reference and search for the item in the P2P network. The *value* contains the payload data.

Nodes in the P2P network offer the following operations to client applications:

- **join(n)** joins the P2P network using another node *n*.
- **leave()** leaves the P2P network (if it is part of the network).
- **store(key,value)** stores data in the P2P network. May be invoked from any active node. It should be able to locate the node "responsible" for the provided data item.
- **lookup(key)** queries for data with the given key. Same properties as store().

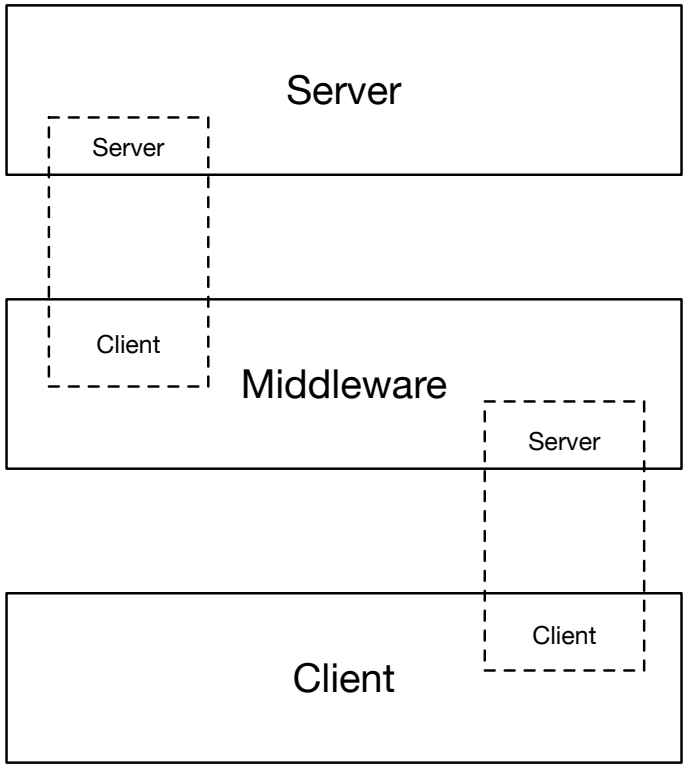
Peer-to-Peer / Chord

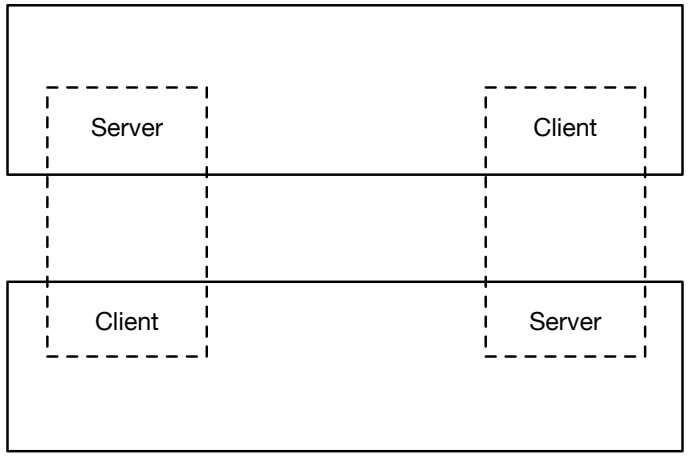
Hand-in: 2024 December 8th (ADAM),
+ interview (Dec 9th)

Modality: 2 people

Peer-to-Peer systems

- distributed, without centralized control or hierarchical organization
- each node has equivalent functionality





Chord

Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications

Ion Stoica; Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan†
MIT Laboratory for Computer Science
chord@lcs.mit.edu
http://pdos.lcs.mit.edu/chord/

Abstract

A fundamental problem that confronts peer-to-peer applications is to efficiently locate the node that stores a particular data item. This paper presents *Chord*, a distributed lookup protocol that addresses this problem. Chord provides support for just one operation: given a key, it maps the key onto a node. Data location can be easily implemented on top of Chord by associating a key with each data item, and storing the key/data item pair at the node to which the key maps. Chord adapts efficiently as nodes join and leave the system, and can answer queries even if the system is continuously changing. Results from theoretical analysis, simulations, and experiments show that Chord is scalable, with communication cost and the state maintained by each node scaling logarithmically with the number of Chord nodes.

1. Introduction

Peer-to-peer systems and applications are distributed systems without any centralized control or hierarchical organization, where the software running at each node is equivalent in functionality. A review of the features of recent peer-to-peer applications yields a long list: redundant storage, permanence, selection of nearby servers, anonymity, search, authentication, and hierarchical naming. Despite this rich set of features, the core operation in most peer-to-peer systems is efficient location of data items. The contribution of this paper is a scalable protocol for lookup in a dynamic peer-to-peer system with frequent node arrivals and departures.

The *Chord protocol* supports just one operation: given a key, it maps the key onto a node. Depending on the application using Chord, that node might be responsible for storing a value associated with the key. Chord uses a variant of consistent hashing [11] to assign keys to Chord nodes. Consistent hashing tends to balance load, since each node receives roughly the same number of keys, and involves relatively little movement of keys when nodes join and leave the system.

Previous work on consistent hashing assumed that nodes were aware of most other nodes in the system, making it impractical to scale to large number of nodes. In contrast, each Chord node needs "routing" information about only a few other nodes. Because the routing table is distributed, a node resolves the hash function by communicating with a few other nodes. In the steady state, in an N-node system, each node maintains information only about $O(\log N)$ other nodes, and resolves all lookups via $O(\log N)$ messages to other nodes. Chord maintains its routing information as nodes join and leave the system; with high probability each such event results in no more than $O(\log^2 N)$ messages.

Three features that distinguish Chord from many other peer-to-peer lookup protocols are its simplicity, provable correctness, and provable performance. Chord is simple, routing a key through a sequence of $O(\log N)$ other nodes toward the destination. A Chord node requires information about $O(\log N)$ other nodes for *efficient* routing, but performance degrades gracefully when that information is out of date. This is important in practice because nodes will join and leave arbitrarily, and consistency of even $O(\log N)$ state may be hard to maintain. Only one piece information per node need be correct in order for Chord to guarantee correct (though slow) routing of queries; Chord has a simple algorithm for maintaining this information in a dynamic environment.

The rest of this paper is structured as follows. Section 2 compares Chord to related work. Section 3 presents the system model that motivates the Chord protocol. Section 4 presents the base Chord protocol and proves several of its properties, while Section 5 presents extensions to handle concurrent joins and failures. Section 6 demonstrates our claims about Chord's performance through simulation and experiments on a deployed prototype. Finally, we outline items for future work in Section 7 and summarize our contributions in Section 8.

2. Related Work

While Chord maps keys onto nodes, traditional name and location services provide a *direct* mapping between keys and values. A value can be an address, a document, or an arbitrary data item. Chord can easily implement this functionality by storing each key/value pair at the node to which that key maps. For this reason and to make the comparison clearer, the rest of this section assumes a Chord-based service that maps keys onto values.

DNS provides a host name to IP address mapping [15]. Chord can provide the same service with the name representing the key and the associated IP address representing the value. Chord requires no special servers, while DNS relies on a set of special root

* University of California, Berkeley. istoica@cs.berkeley.edu
† Authors in reverse alphabetical order.

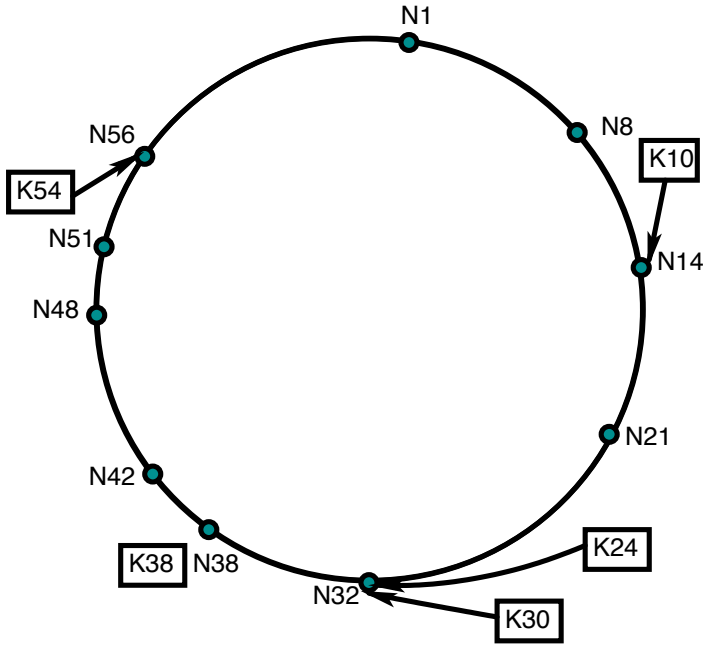
This research was sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Space and Naval Warfare Systems Center, San Diego, under contract N66001-00-1-8933.

SIGCOMM'01, August 27-31, 2001, San Diego, California, USA.

Stoica et al., "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications". In: Proc. SIGCOMM, 2001, pp. 149–160.

(downloadable from within the university network)

Chord



- `store(caller, key, value)` allows to store data in the P2P network (callable from any node in the network).
- `lookup(caller, key)` allows to query for a data item with a given key (callable from any node in the network).
- `delete(caller, key)` allows to remove a data item with a given key (callable from any node in the network).

Exercise 7

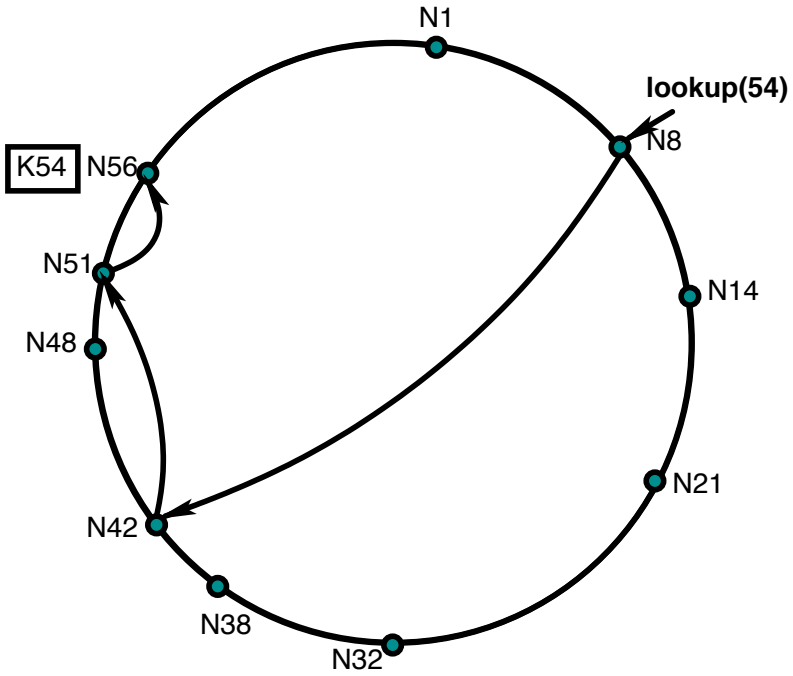
Goal: Understand how Chord works and be able to compare it to the other P2P systems you have seen in the lecture.

Suggestions

- Read the instructions on the exercise sheet carefully (!)
- Have a look at the provided source code (Java), which you are required to complete (`ChordPeer` class).
- Read the SIGCOMM Chord paper carefully (!)
- Understand the steps required to implement for the Chord network (see pseudo-code)

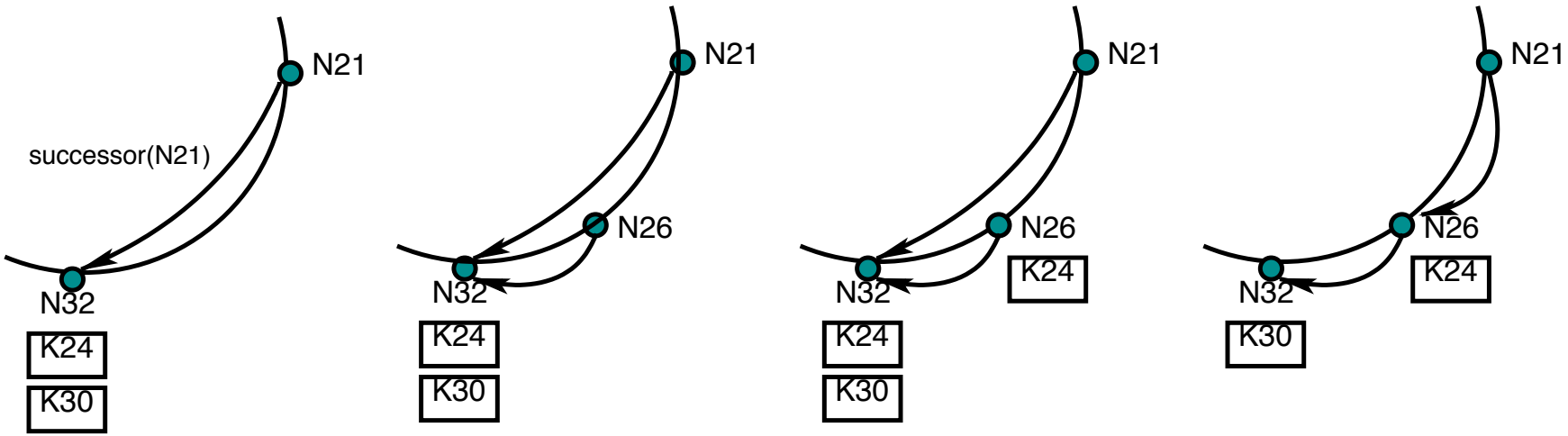
Important: All the required material (exercise sheet, Chord paper, source code) can be found on ADAM.

Question 1: Basic Chord Network



- Network primitives (**sucessor**, **predecessor** etc.)
- Lookup using finger table (i.e. routing of requests)
- Handle nodes joining the network

Question 2: Dynamic Joining/Departing



- Handle dynamic and continuous update of finger tables as nodes join.
- Handle nodes leaving the network.

Simulation framework

We provide you with a simulation framework (for) which:

- you have to complete the implementation (`ChordPeer` class).
- allows you to simulate the network in the two scenarios (static vs. dynamic).
- provides you with some basic functionality (hash functions, modulo arithmetic etc.)

Important: We recommend you to use **Java 17** (Open JDK), for which you should be able to use the framework out-of-the-box. However, if you want to use it with another Java version, it should be possible, but you must make some adjustments to the dependencies (Gradle). See Readme file for further instructions!

Interview

During the interview, you will be asked questions related to the topic of the exercise (P2P, Chord, etc.) and your specific solutions.

- 15 min, scheduled meeting (Xoyondo)
- Understanding of the code **and** the topic expected.
- Each group member has to attend the meeting.
- All team members must be able to answer the questions.
- Each team member has to register.

Register for the interview (“first come first served”) on

`https://xoyondo.com/dp/pzedfa6cfg99hd5`

(Remember: Each team member has to register on their own!)

Questions?

yiming.wu@unibas.ch