Introduction to Distributed Systems & Databases

COS 316 Lecture 2

Amit Levy

Course Administrativia

Warning #1: This is a new course

Warning #2: System Building is nt *just* Programming

- COS126 & 217 told you how to design & structure your programs.
- This class doesn't.
- If your system is designed poorly it can be much harder to get right!
- Converseley, assignments won't require algorithms or data structures you're not already familiar with.
    - 4xx systems classes require both!
- Your friends:
    - Working in teams (don't worry, you're *required to*)
    - Discussing potential solutions before implementing
    - Test-driven development

Learning Objectives: Skills

- Go programming language, and “Systems” programming
- Version control with git
- Working in groups
- “Systems programming”:
    - sockets programming, concurrency, modular design, unit testing, performance measurement...

Learning Objectives: System Design Princeiples

- What is the field of systems?
  - Learn to appreciate the trade-offs in designing and building the systems you use.
  - Get better at understanding how systems work.
  - Learn to *use* systems better—write more efficient/secure/robust/etc applications.
- 4 major themes:
  1. Naming (weeks 3 & 4)
  2. Caching (weeks 5 & 6)
  3. Resource Management (weeks 7 & 8)
  4. Virtualization (weeks 9 & 10)
  5. Access Control (weeks 11 & 12)

Lectures

- Each two week unit explores a different cross-cutting theme
- Each two week unit will use one or two systems to explore the theme

Workload

- 50% Programming assignments – due every other week
- 10% Problem sets – every other week
- 10% Dean’s day written project
- 30% Two Midterms
- Occassional readings

Assignments: We're building a web framework

Each of the assignments has you building a component of a web framework.

Assignments: We're building a web framework

Each of the assignments has you building a component of a web framework.

A web framework is a set of libraries and tools for building complex web applications:

- Abstracts connection and protocol handling
- Routes requests to controllers/handlers
- Caching for common queries and computations
- Multiplexes concurrent access to databases
- Translates database objects into programming language constructs
- User authentication and authorization

Assignments: We're building a web framework

Each of the assignments has you building a component of a web framework.

A web framework is a set of libraries and tools for building complex web applications:

- Abstracts connection and protocol handling
- Routes requests to controllers/handlers
- Caching for common queries and computations
- Multiplexes concurrent access to databases
- Translates database objects into programming language constructs
- User authentication and authorization

Examples: Rails, Django, Express, Apache Struts, Laravel

Assignments: Collaboration & Resources *This slide is really important*

- You can, and *should* use any resources available on the Internet to complete assignments:
  - Go documentation, Stackoverflow, open source projects
  - Mailing lists, chat rooms, etc...
  - Cite sources in your comments or README!
- You *must* collaborate (in groups of 2)
- You may *not* ask instructors for help debugging your code.
- ~~*Gilligan's Island*~~ *Game of Thrones* rule:
  - If you discuss the assignment with other teams, do something else for an hour before returning to your code.

Assignments: Submitting & Grading

- Submitting happens whenever you “push” to your “master” branch on GitHub
  - You can push as many times as you’d like (I encourage you to do so *often*)
- Grading is automatic and immediate
  - There is no penalty for multiple submissions. We will use your highest graded submission
  - Each automatic grading is posted as a comment to the last commit of each push. It includes a break down of tests cases, including which failed.
- Late days:
  - 7 days total for the semester
  - Assigned retroactively to give you the best possible overall grade

Late days example

1. Parker submits assignment #1 on time, but can't figure out how to pass the last test case. Their grade so far for the assignment is 95%.
2. 4 days after the deadline, Parker figures out how to pass the last test and submits late, getting 100%.
3. Months later... Parker underestimates their workload and isn't able to submit assignment 4 until 4 days after the deadline, but passes all tests to get 100%.

Late days example

1. Parker submits assignment #1 on time, but can't figure out how to pass the last test case. Their grade so far for the assignment is 95%.
2. 4 days after the deadline, Parker figures out how to pass the last test and submits late, getting 100%.
3. Months later... Parker underestimates their workload and isn't able to submit assignment 4 until 4 days after the deadline, but passes all tests to get 100%.
4. We assign the late days to assignment 4, so that Parker's grade is 90% + 100%, as opposed to 90% + 0%.

Questions?

Preview of Distributed Systems

What is a distributed system?

More than one computer working together to solve a “systems” problem, e.g.:

- Storage and delivery
- Computation
- Coordination or agreement

Examples:

- MapReduce - run computations over very large data sets

What is a distributed system?

More than one computer working together to solve a “systems” problem, e.g.:

- Storage and delivery
- Computation
- Coordination or agreement

Examples:

- MapReduce - run computations over very large data sets
- Content Distribution Network (CDN) - delivers web pages close to users

What is a distributed system?

More than one computer working together to solve a “systems” problem, e.g.:

- Storage and delivery
- Computation
- Coordination or agreement

Examples:

- MapReduce - run computations over very large data sets
- Content Distribution Network (CDN) - delivers web pages close to users
- Paxos (or VSR, or Raft) - Coordinate “consensus” on failure resiliant decisions

What is a distributed system?

More than one computer working together to solve a "systems" problem, e.g.:

- Storage and delivery
- Computation
- Coordination or agreement

Examples:

- MapReduce - run computations over very large data sets
- Content Distribution Network (CDN) - delivers web pages close to users
- Paxos (or VSR, or Raft) - Coordinate "consensus" on failure resiliant decisions
- *BitCoin* - coordinate the mass consumption of fossil fuels to gain currency with no *real* value and tenuous economic value

Synthesize: When do we need systems with more than one computer?

Lots of computation

Fault tolerance

Lots of storage or memory

Mutiple people access same resource

Synthesize: When do we need systems with more than one computer?

- Fault tolerance
- Not enough resources on a single machine
- Accomplish something faster
- Alleviate stress from scarce resource

Consistent Hashing with Chord

Problem statement

We want to store more data than can fit on a single machine… or even a single data center… Or we want to use end-user run computers…

Problem statement

We want to store more data than can fit on a single machine... or even a single data center... Or we want to use end-user run computers...

The year is 2000...

- Radiohead release *Kid A*, their first top-20 record
- The dot-com bubble bursts
- Napster is being sued and might shut down

Problem statement

We want to store more data than can fit on a single machine… or even a single data center… Or we want to use end-user run computers…

The year is 2000…

- Radiohead release *Kid A*, their first top-20 record
- The dot-com bubble bursts
- Napster is being sued and might shut down

We want a system with no central coordination that can store lots of data, easily accessible by anyone.

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

Ion Stoica[†], Robert Morris[‡], David Liben-Nowell[‡], David R. Karger[‡], M. Frans Kaashoek[‡], Frank Dabek[‡], Hari Balakrishnan[‡]

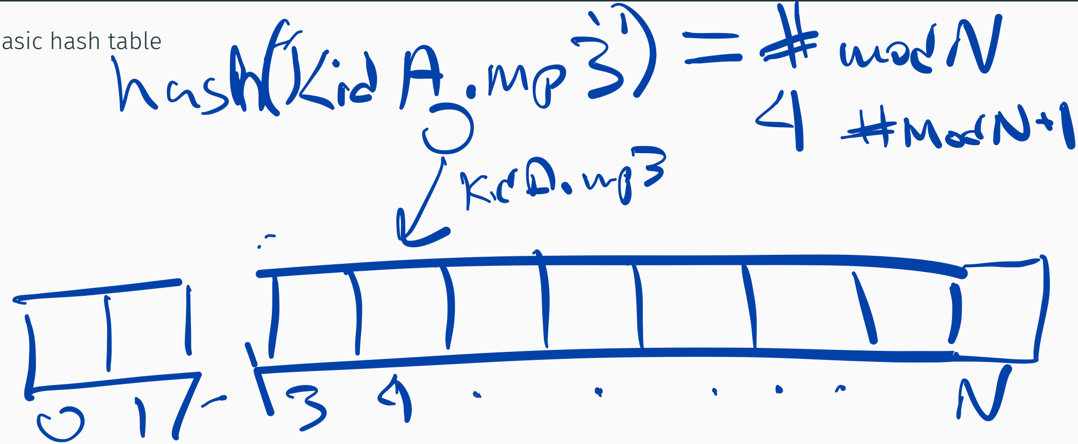
Abstract—

A fundamental problem that confronts peer-to-peer applications is the efficient location of the node that stores a desired 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 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 and simulations show that Chord is scalable: communication cost and the state maintained by each node scale logarithmically with the number of Chord nodes.

tem.

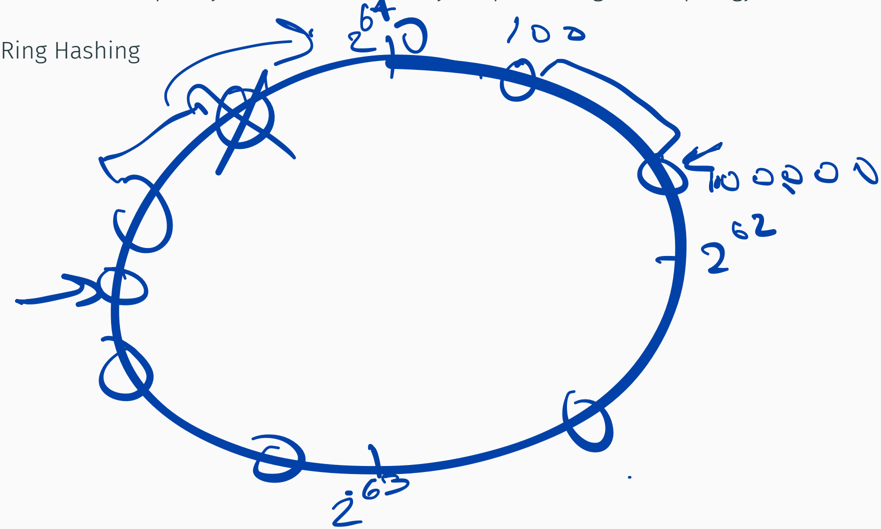
A Chord node requires information about $O(\log N)$ other nodes for *efficient* routing, but performance degrades gracefully when that information is out of date. This is important in practice because nodes will join and leave arbitrarily, and consistency of even $O(\log N)$ state may be hard to maintain. Only one piece of information per node need be correct in order for Chord to guarantee correct (though possibly slow) routing of queries; Chord has a simple algorithm for maintaining this information in a dynamic environment

A basic hash table



Solution? Consistent Hashing

- A hash function that maps keys to bins consistently despite changes in topology.
- Example: Ring Hashing



How much information to do a lookup?

If N nodes in the ring?

$O(N)$

Is this reasonable?

What if `N` is in the millions?

- Storage?
- Discovery?
- Churn?

Solution: Trade Lookups for Local Storage

- Each Chord node keeps track of k other "finger" nodes:
    - $finger[k] = NodeFor(n + 2^{k-1} mod 2^m, 1 <= k <= m)$
    - $successor = finger[1]$

```
// ask node n to find the successor of id
def n.find successor(id):
  if (n < id && id <= successor)
    return successor;
  else
    n* = closest_preceding_node(id);
    return n*.find_successor(id);

// search the local table for the highest predecessor of id
def n.closest_preceding_node(id):
  for i = m downto 1
    if (n < finger[i] && finger[i] < id)
      return finger[i];
  return n;
```

$O(\log N)$

Is this reasonable?

- Storage?
- Discovery?
- Churn?

Up next

Assignment 1 released

- Due 9/24 @ 11pm

Two short readings (posted on the website)

- The Rise of Worse is Better
- Worse is Better is Worse

Next time: Introduction to Security