Week 13 – Wednesday

COMP 3400

Last time

- What did we talk about last time?
- Limits of parallelism
  - Amdahl's Law
  - Gustafson's Law
- Timing in distributed environments
  - Clock synchronization
  - Lamport timestamps
  - Vector clocks

Questions?

Assignment 7

Reliable Storage and Location

Reliable data storage

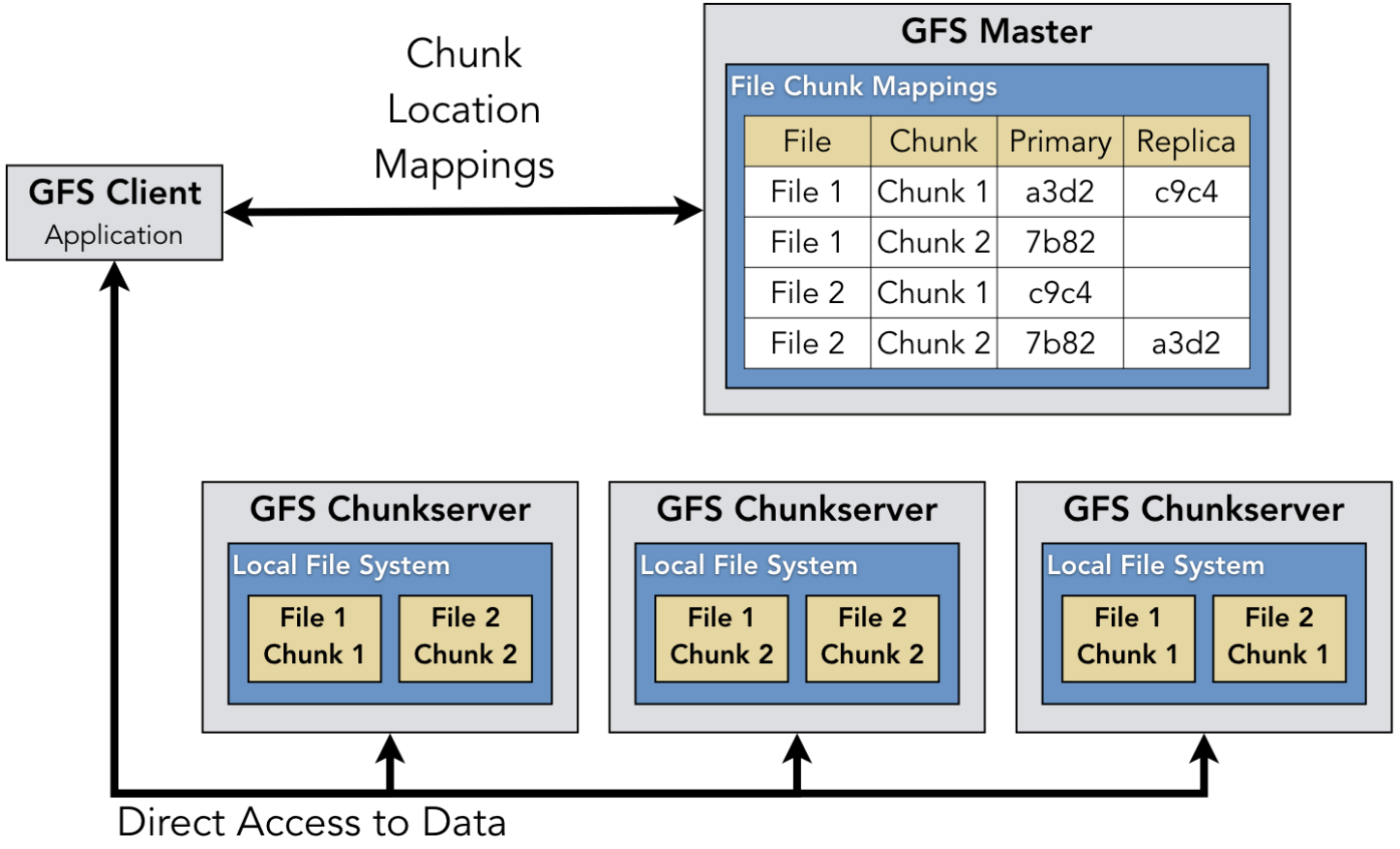
- If you want to get a file from a web server, you can go to a URL and make an HTTP request
- Unfortunately, if that server is down or unreachable, you can't get the file
- For this reason, distributed systems are often used to store data
- A key feature of distributed data storage is **replication**, keeping multiple copies of the same data
  - Replication avoids a single point of failure
  - If done correctly, replication can also do load balancing, improving performance by providing multiple sources for data

Google File System

- The Google File System (GFS) is a distributed storage system
- GFS was designed to store Google's internal data, like the data structures used for PageRank
- Files are often large, so they're broken into chunks
- Chunks are stored on chunkservers as regular files
- A master server stores a table mapping file chunks to their locations

Illustration of GFS

- Each chunk has a primary chunkserver as well as replicas
- The chunks are identical, but the primary chunkserver is the only place where the chunk can be modified
  - It propagates changes to the other chunkservers
  - This redundancy makes writing to GFS slower, even though reading is relatively fast
- The master server periodically sends messages to the chunkservers to get their current status

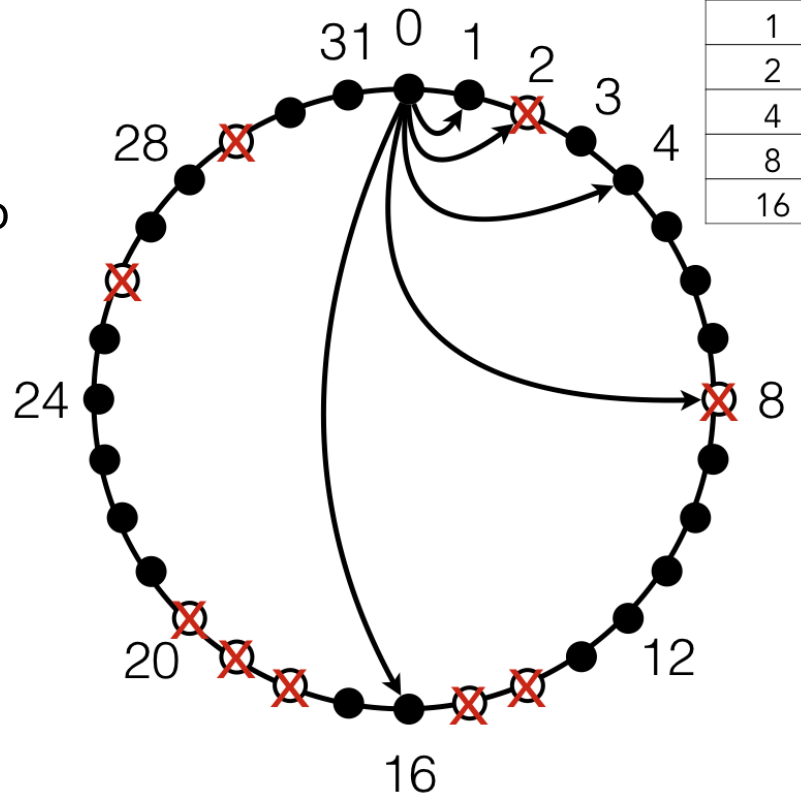


Distributed hash tables

- GFS was designed by Google for its own purposes
  - It uses a central server
  - Servers keep information about each other
- What if we have no idea what servers are going to be in the network?
- **Distributed hash tables (DHT)** are an approach for mapping arbitrary objects to arbitrary servers
- DHTs are a way to organize a peer-to-peer network to avoid query flooding

Chord DHT

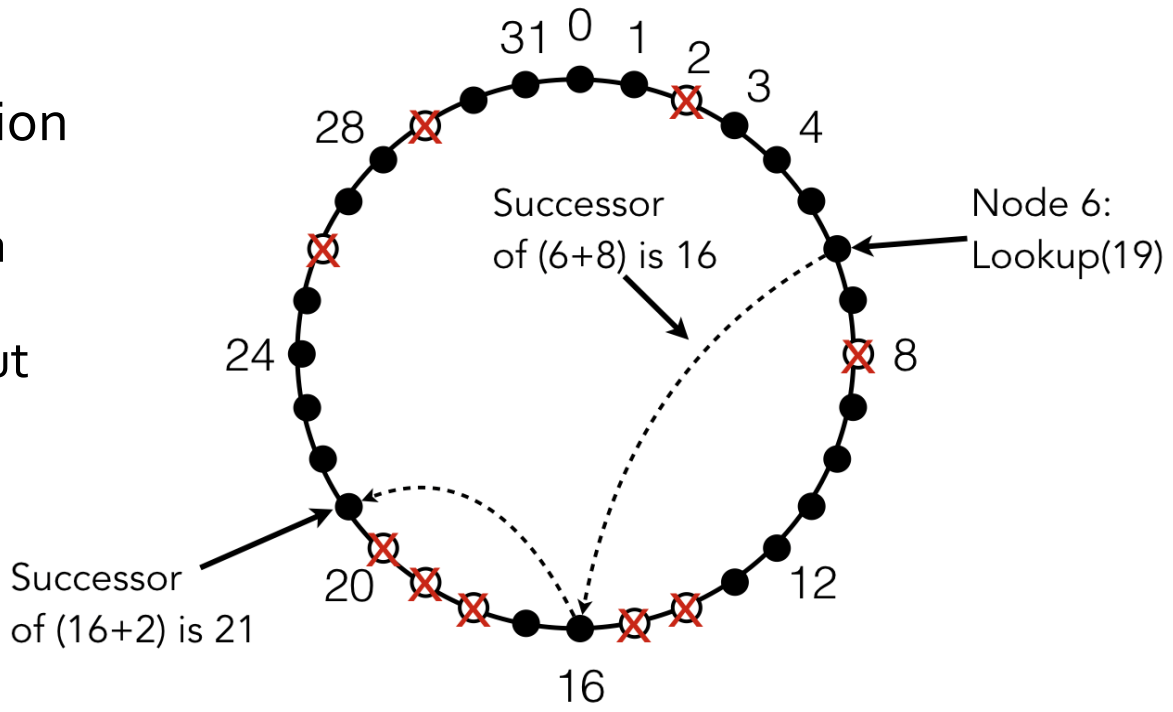
- Chord was one of the first algorithms for a DHT, introduced in 2001
- Each node has a unique identifier (often its IP address) that's hashed to provide a location in a circle
  - If the hash is ***n*** bits long, the DHT can support up to 2^n nodes
- Most locations in the circle are empty
- Each node has a "finger table," tracking successor elements in increasing powers of 2 away on the circle
  - If the power of 2 node is missing, it tracks the next non-missing node
- The example on the right is only for $2^5 = 32$ nodes



Hop	Successor(0+hop)
1	1
2	3
4	5
8	9
16	16

Files in Chord DHT

- When a file is added, it's hashed
- Whichever node has that hash value (or is its successor) is the location of that file
- On the right, node 6 is looking for a file at location 19 (the successor of 18)
  - It looks at 6 + 8 = 14, which doesn't exist but has a successor of 16
  - Then it looks at 16 + 2 = 18, which doesn't exist but has a successor of 21
  - Node 21 is where the file is supposed to be
- The details get a little more complex, but the practical result is that a file can be found with O(log ***n***) requests, where ***n*** is the size of the network
- Replication is done by caching files at nodes that were part of the lookup to find the file



Consensus in Distributed Systems

State machines

- At a high level, you can think of distributed systems as state machines
- The system contains many variables with different values that change over time
  - The idea is broader than a finite state machine, since the system is more complex than having a single state
- In a distributed system, different nodes could have different internal representations of these values
- If the nodes agree on a state variable's value, they have **consensus**

Consensus

- Reaching consensus is the goal of many distributed protocols
- To reach consensus, a protocol must have three properties:
  - **Termination:** Every correct (non-failing) process will eventually decide on a value
  - **Integrity:** If every correct process proposes the same value, any correct process must decide that value
  - **Agreement:** All correct processes decide the same value
- Examples
  - In GFS, a consensus protocol could tell any node whether a particular file was on a particular node
  - In NTP, nodes will be able to agree on synchronized time

Failure

- However, processes **do** fail in distributed systems
- Failure could mean making some error, crashing, going into an infinite loop, or losing connection to the network
  - Processes could even be malicious, trying to undermine the system
- Even in the face of (many?) failures, we'd like the distributed system to reach consensus
- A common analogy used to describe this problem is the Byzantine generals problem

Byzantine generals

- The Byzantine generals problem imagines three generals stationed around a large city
  - Only if all three generals decide to attack, they can defeat the city
  - If all three generals decide to retreat, they can retreat with minimal casualties
  - But if some attack and some retreat, they're all going to get slaughtered
- The generals exchange messages with each other to see what they want to do
- What if one general tells A that he's going to retreat and B that he's going to attack?
  - A will decide to retreat but B will go ahead and attack

More on Byzantine generals

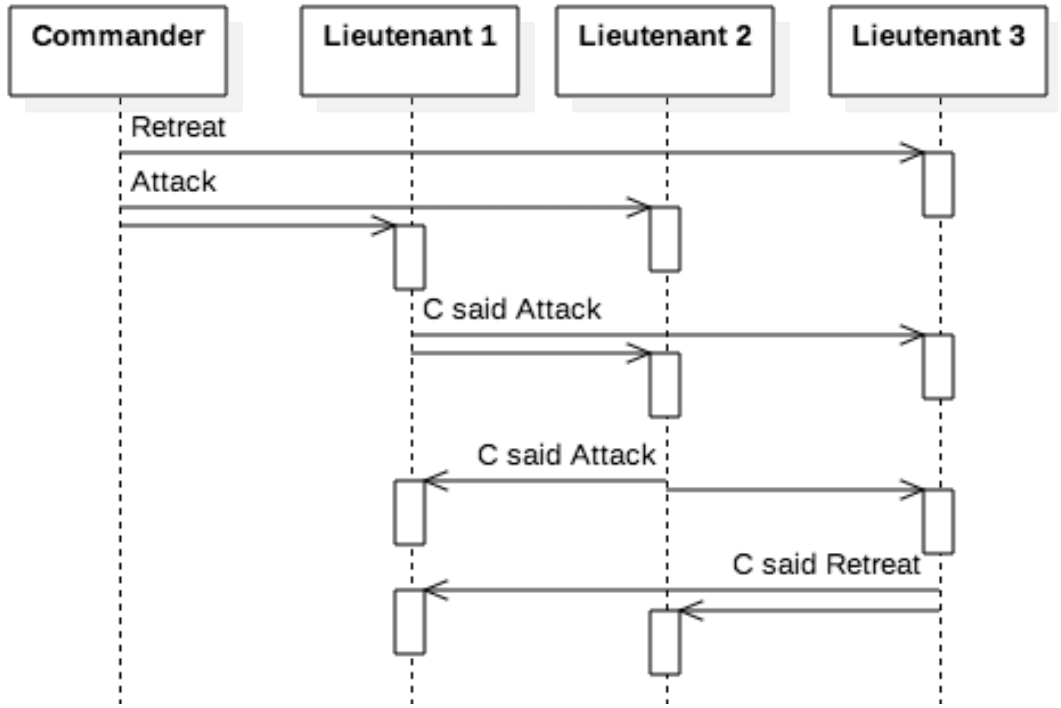
- A different (but equivalent) version of the Byzantine generals problem imagines:
  - One general is the commander who decides what to do
  - The other two are lieutenants who check with each other to make sure that they got the same message from the commander
- What if there's one bad general?
  - A bad commander could send retreat to one lieutenant and attack to the other
  - A bad lieutenant could receive attack from the commander but send retreat to the other lieutenant
  - A good lieutenant couldn't distinguish between those two situations

Limits on consensus

- Consensus is hard
  - Failing processes can mess things up for correct processes
  - Because there's limited information, a process can appear to be correct to one part of the system and failing to another
- A **Byzantine failure** is exactly this kind
  - There's conflicting information, and it's impossible to determine what's reliable

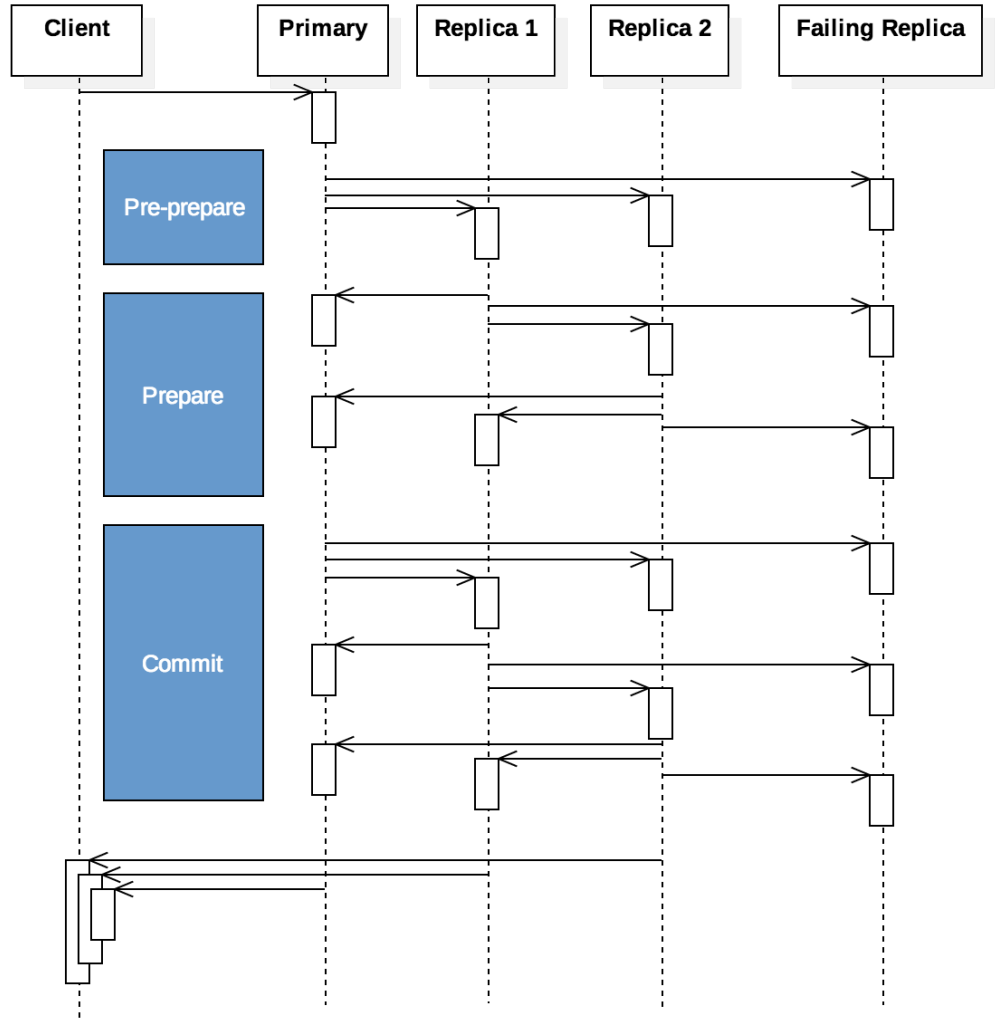
The 1/3 limit

- It's not an accident that the number of generals chosen is three
- If strictly less than 1/3 of the nodes are failing, it's possible to achieve consensus
- If we extend the problem to four generals (three lieutenants), then generals who are working can decide on a consensus using majority rule
- Even a bad commander who issues confusing orders won't mess up the system
- However, knowing what the consensus is doesn't tell us which nodes are failing
- Also, this 1/3 limit depends on synchronous communication



Practical Byzantine fault tolerance

- Consensus is a pretty high bar
- Practical Byzantine fault tolerance (PBFT) tries to make a workable system with consistency among nodes without requiring consensus
  - A client asks a primary process for some data
  - The primary process asks replicas for the data
  - The client considers the majority response to be correct after getting enough responses
- It works because
  - Messages are cryptographically signed
  - Messages are numbered so that old messages won't be reused
  - Responses have a time limit
- We assume that less than 1/3 of the processes are faulty, meaning that if we get messages from more than 1/3 of the processes that agree, we assume the response is good



Blockchains

Blockchains

- Blockchains are a form of distributed ledger
- Unlike banks, which are centralized authorities for transactions that have occurred at the banks, blockchains try to record transactions in a distributed way with *no* central authority
- Although similar ideas existed before, blockchains as we know them were invented by Satoshi Nakamoto (real identity unknown) in 2008
- The original blockchain idea was intended to keep track of bitcoin transactions
- Now, most cryptocurrencies use some form of blockchain to track transactions

Double-spending

- Blockchains are distributed systems that can be used to record almost anything
- But their use has been dominated by cryptocurrency
- A central problem that any digital money faces is **double-spending**
  - What stops someone from spending a digital token more than once?
- Transactions are recorded in blockchains
- Two competing blockchains could record different transactions, but the longer chain is considered the valid one

Proof-of-work

- Blockchains are built by recording transactions along with other data that hashes in a specified pattern
  - Usually, a hash value with a certain number of zeroes at the beginning
- It's easy to check that the transaction has the right hash value
- But it's computationally difficult to generate data that has a hash with a certain number of zeroes at the beginning
- And that's what mining is: Trying random strings until something has the right hash value
- Since a large number of strings will have the right hash value, an entity with more than 50% of the computational power working on a blockchain network could outpace everyone else writing transactions, taking control of it

Why blockchains?

- Blockchains are trying to solve the same problem we talked about with Byzantine generals: consensus
- However, the goal isn't to retrieve data efficiently
- The goal is to make a record that's hard to dispute
  - Even if there are malicious actors who want to lie about the transactions
- Block time is how long it takes for the network to add a block, recording a set of transactions
  - About 12 seconds in Ethereum
  - About 10 minutes in Bitcoin
- Blockchains can also be used to record ownership of digital items
  - Like the non-fungible tokens (NFTs) that people went crazy over for a few months

Why not blockchains?

- Most current blockchains use **proof-of-work** to show that the blocks are valid
  - Ethereum switched over to proof-of-stake, which is estimated to consume only 0.005% of the power needed by Bitcoin
- Proof-of-work requires huge computational effort to find a string whose hash starts with enough zeroes
- A 2024 review from the Institut Polytechnique de Paris estimates that Bitcoin mining uses about the same power as Poland
- Best estimates suggest that a single Bitcoin transaction uses hundreds of thousands as times as much energy as a Visa transaction
- Blockchains are often structured to require more computational effort when more people are mining
- There's a real environmental cost as well as the use of electricity that could be used for practical things

Ticket Out the Door

Upcoming

Next time…

- Review up to Exam 1

Reminders

- Work on Assignment 7
  - **Due Thursday before midnight!**
- **No class on Friday!**