Exam Review

CS 475, Fall 2019
Concurrent & Distributed Systems
Course Topics

• This course will teach you **how** and **why** to build distributed systems
• Distributed System is “a collection of independent computers that appears to its users as a single coherent system”
• This course will give you theoretical knowledge of the tradeoffs that you’ll face when building distributed systems
Course Topics

How do I run multiple things at once on my computer?
Concurrency, first half of course

How do I run a big task across many computers?
Distributed Systems, second half of course
Concurrency

• Goal: do multiple things, at once, coordinated, on one computer
  • Update UI
  • Fetch data
  • Respond to network requests
  • Improve responsiveness, scalability
• Recurring problems:
  • Coordination: what is shared, when, and how?
Why expand to distributed systems?

- Scalability
- Performance
- Latency
- Availability
- Fault Tolerance
More machines, more problems

• More machines -> more chance of seeing at least one machine fail
• PLUS, the network may be:
  • Unreliable
  • Insecure
  • Slow
  • Expensive
  • Limited
Constraints

• Number of nodes
• Distance between nodes
Constraints

- Number of nodes
- Distance between nodes

Even if cross-city links are fast and cheap (are they?)
Still that pesky speed of light…
Recurring Solution #1: Partitioning

All accesses go to single server
Recurring Solution #1: Partitioning

- Divide data up in some (hopefully logical) way
- Makes it easier to process data concurrently (cheaper reads)

Each server has 50% of data, limits amount of processing per server.

Even if 1 server goes down, still have 50% of the data online.
Recurring Solution #2: Replication

All accesses go to single server
Recurring Solution #2: Replication

Entire data set is copied
Recurring Solution #2: Replication

• Improves performance:
  • Client load can be evenly shared between servers
  • Reduces latency: can place copies of data nearer to clients
• Improves availability:
  • One replica fails, still can serve all requests from other replicas
Partitioning + Replication
Partitioning + Replication

- A [0..100] B [A..N]
- A [0..100] B [A..N]
- A [0..100] B [A..N]
Partitioning + Replication

DC

NYC

SF

London
Recurring Problem: Replication

- Replication solves some problems, but creates a huge new one: consistency

Set A=5

“OK”!

Read A

“6”!

5

7

6

7

OK, we obviously need to actually do something here to replicate the data… but what?
Sequential Consistency

Set A = 5

"OK"!

Read A

"5"

Set A = 5

"OK!"

"OK!"
Availability

- Our protocol for sequential consistency does NOT guarantee that the system will be available!
Consistent + Available

Set A=5

"OK"!

Read A

"5"!

Set A=5

Assume replica failed

5 7

6 7
Still broken...

Set $A=5$

“OK”!

Read $A$

“6”!

Assume replica failed
Network Partitions

- The communication links between nodes may fail arbitrarily
- But other nodes might still be able to reach that node

Set A=5 → "OK"!
Read A → "6"!
Assume replica failed

Set B=5
Byzantine Faults

Set A=5

“OK”!

Read A

“6”!

Set A=5

“OK!”

5 7

6 7
CAP Theorem

• Pick two of three:
  • Consistency: All nodes see the same data at the same time (strong consistency)
  • Availability: Individual node failures do not prevent survivors from continuing to operate
  • Partition tolerance: The system continues to operate despite message loss (from network and/or node failure)
• You can not have all three, ever*
  • If you relax your consistency guarantee (we’ll talk about in a few weeks), you might be able to guarantee THAT…
CAP Theorem

• C+A: Provide strong consistency and availability, assuming there are no network partitions
• C+P: Provide strong consistency in the presence of network partitions; minority partition is unavailable
• A+P: Provide availability even in presence of partitions; no strong consistency guarantee
Agreement Generally

• Most distributed systems problems can be reduced to this one:
  • Despite being separate nodes (with potentially different views of their data and the world)…
  • All nodes that store the same object O must apply all updates to that object in the same order (consistency)
  • All nodes involved in a transaction must either commit or abort their part of the transaction (atomicity)
• Easy?
  • … but nodes can restart, die or be arbitrarily slow
  • … and networks can be slow or unreliable too
Properties of Agreement

- **Safety** (correctness)
  - All nodes agree on the same value (which was proposed by some node)
- **Liveness** (fault tolerance, availability)
  - If less than N nodes crash, the rest should still be OK
2PC Example

If we can commit, then lock our customer, vote “yes”

If everyone can commit, then outcome == commit, else abort

Coordinator (client or 3rd party)  
transaction .commit()  
prepare  
responseGNB  
responseD&D  
outcome

Participant Goliath National  
prepare

Participant Duke & Duke  
prepare

outcome
3 Phase Commit

- Goal: Avoid blocking on node failure
- How?
  - Think about how 2PC is better than 1PC
    - 1PC means you can never change your mind or have a failure after committing
    - 2PC **still** means that you can’t have a failure after committing (committing is irreversible)
  - 3PC idea:
    - Split commit/abort into 2 sub-phases
      - 1: Tell everyone the outcome
      - 2: Agree on outcome
    - Now: EVERY participant knows what the result will be before they irrevocably commit!
Partitions

Implication: if networks can delay arbitrarily, 3PC does not guarantee safety!!!!

Timeout behavior: abort

Coordinator

Soliciting Authorized

Prepared to commit

Network Partition!!!

Yes
Participant A
Committed

Yes
Participant B
Committed

Yes
Participant C
Aborted

Yes
Participant D
Aborted

Implication: if networks can delay arbitrarily, 3PC does not guarantee safety!!!!

Timeout behavior: abort

Commit!

Timeout behavior: commit!

Uncertain

Uncertain

Uncertain

Uncertain
Can we fix it?

- Short answer: No.
- Fischer, Lynch & Paterson (FLP) Impossibility Result:
  - Assume that nodes can only fail by crashing, network is reliable but can be delayed arbitrarily
  - Then, there can not be a deterministic algorithm for the consensus problem subject to these failures
FLP - Intuition

• Why can’t we make a protocol for consensus/agreement that can tolerate both partitions and node failures?
• To tolerate a partition, you need to assume that eventually the partition will heal, and the network will deliver the delayed packages
• But the messages might be delayed forever
• Hence, your protocol would not come to a result, until forever (it would not have the liveness property)
Domain Name System

- **Global Layer**
  - net
  - org
  - edu
  - com
  - gov
  - uk

- **Administration Layer**
  - root-servers
  - gmu

- **Managerial Layer**
  - www
  - www
  - cs
  - www
NFS Caching - Close-to-open

1. Open File
2. Read File: “a”

3. Open File
4. Write File: “b”

5. Open File
6. Read File: “a”

7. Close File

8. Open File
9. Read File: “b”

Note: in practice, client caches periodically check server to see if still valid
GFS Architecture
ZooKeeper - Guarantees

- **Liveness guarantees**: if a majority of ZooKeeper servers are active and communicating the service will be available.
- **Durability guarantees**: if the ZooKeeper service responds successfully to a change request, that change persists across any number of failures as long as a quorum of servers is eventually able to recover.
Hadoop + ZooKeeper

ZK Server

NameNode

DataNode

ZK Client

timeout

disconnected

Notification that leader is gone, secondary becomes primary

NameNode

ZK Client

DataNode

Primary

Secondary

Primary

Secondary

DataNode

DataNode

DataNode

DataNode

DataNode

DataNode

DataNode

DataNode

DataNode
Can I have f1?

where is f1?

client 1

c2 has f1

client 2

where is f1?

client 3

client 4

client 5
Example Threat: Web Server

Might be “man in the middle” that intercepts requests and impersonates user or server.

HTTP Request

client page (the “user”)

malicious actor “black hat”

Do I trust that this request \textit{really} came from the user?

Do I trust that this response \textit{really} came from the server?
Sample Questions and Discussion - Socrative

Go to socrative.com and select “Student Login” Room: CS475; ID is your G-Number

Reminder: If you are not in class, you may not complete the activity. If you do anyway, this will constitute a violation of the honor code.