Client–Server Model

- Client makes request to server
- Server acts and responds

(For example: Email, WWW, Printing, etc.)

Client–Server: Three-Tier Server

Peer-to-Peer: Unstructured

(For example: Kazaa, Gnutella)

Peer-to-Peer: Structured (DHT)

- Circular DHT:
  - Only aware of neighbours
  - O(n) lookups

- Implement shortcuts
  - Skips ahead
  - Enables binary-search-like behaviour
  - O(log(n)) lookups
Desirable Criteria for Distributed Systems

- **Transparency:**
  - Appears as one machine
- **Flexibility:**
  - Supports more machines, more applications
- **Reliability:**
  - System doesn’t fail when a machine does
- **Performance:**
  - Quick runtimes, quick processing
- **Scalability:**
  - Handles more machines/data efficiently

Eight Fallacies (to avoid)

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn’t change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

LIMITATIONS OF DISTRIBUTED COMPUTING: CAP THEOREM

But first ... ACID

Have you heard of ACID guarantees in a database class?

For traditional (non-distributed) databases ...

1. **Atomicity:**
   - Transactions all or nothing: fail cleanly
2. **Consistency:**
   - Doesn’t break constraints/rules
3. **Isolation:**
   - Parallel transactions act as if sequential
4. **Durability**
   - System remembers changes

What is CAP?

Three guarantees a distributed sys. could make

1. **Consistency:**
   - All nodes have a consistent view of the system
2. **Availability:**
   - Every read/write is acted upon
3. **Partition-tolerance:**
   - The system works even if messages are lost

A Distributed System (Replication)
There's 891 users in 'M'.

There's 891 users in 'M'.

How many users start with 'M'.

How many users start with 'M'.

How many users start with 'M'.

How many users start with 'M'.

How many users start with 'M'.

How many users start with 'M'.

How many users start with 'M'.

How many users start with 'M'.

How many users start with 'M'.

How many users start with 'M'.

Consistency

Availability

Partition-Tolerance

The CAP Question

Can a distributed system guarantee **consistency** (all nodes have the same up-to-date view), **availability** (every read/write is acted upon) and **partition-tolerance** (the system works even if messages are lost) at the same time?

What do you think?

The CAP Answer

The CAP “Proof”
The CAP “Proof” (in boring words)

- Consider machines $m_1$ and $m_2$ on either side of a partition:
  - If an update is allowed on $m_2$ (Availability), then $m_1$ cannot see the change: (loses Consistency)
  - To make sure that $m_1$ and $m_2$ have the same, up-to-date view (Consistency), neither $m_1$ nor $m_2$ can accept any requests/updates (lose Availability)
  - Thus, only when $m_1$ and $m_2$ can communicate (lose Partition tolerance) can Availability and Consistency be guaranteed

The CAP Theorem

A distributed system cannot guarantee consistency (all nodes have the same up-to-date view), availability (every read/write is acted upon) and partition-tolerance (the system works even if messages are lost) at the same time.

(“Proof” as shown on previous slide 😊)
How many users start with ‘M’? There’s 891 users in ‘M’.

In what way was Twitter operating under BASE-like conditions?

• Basically Available
  – Pretty much always “up”

• Soft State
  – Replicated, cached data

• Eventual Consistency
  – Stale data tolerated, for a while

The CAP Theorem

• C,A in CAP ≠ C,A in ACID

• Simplified model
  – Partitions are rare
  – Systems may be a mix of CA/CP/AP
  – C/A/P often continuous in reality!

• But concept useful/frequently discussed:
  – How to handle Partitions?
    • Availability? or
    • Consistency?

Fault Tolerance / Consensus

Synchronous vs. Asynchronous

• Synchronous distributed system:
  – Messages expected by a given time
    • E.g., a clock tick
  – Missing message has meaning

• Asynchronous distributed system:
  – Messages can arrive at any time
    • Delay is finite but not known
  – Missing message could arrive any time!

Faults
Lunch Problem

10:30 AM. Alice, Bob and Chris work in the same city. All three have agreed to go downtown for lunch today but have yet to decide on a place and a time.

Bob

Alice

Chris

Asynchronous Consensus: Texting

10:45 AM. Alice tries to invite Bob for lunch …

Hey Bob, Want to go downtown to McDonald’s for lunch at 12:00AM?

11:35 AM. No response. Should Alice head downtown?

Asynchronous Consensus: Texting

10:45 AM. Alice tries to invite Bob for lunch …

Hey Bob, Want to go downtown to McDonald’s for lunch at 12:00AM?

11:42 AM. No response. Where should Bob go?

Asynchronous Consensus

• Impossible to guarantee!
  -- A message delay can happen at any time and a node can wake up at the wrong time!
  -- Fischer-Lynch-Patterson (1985): No consensus can be guaranteed if there is even a single failure

• But asynchronous consensus can happen
  -- As you should realise if you’ve ever successfully organised a meeting by email or text ;)

Asynchronous Consensus: Texting

10:45 AM. Alice tries to invite Bob for lunch …

Hey Bob, Want to go downtown to McDonald’s for lunch at 12:00AM?

11:38 AM. No response. Did Bob see the acknowledgement?

Asynchronous Consensus: Texting

10:45 AM. Alice tries to invite Bob for lunch …

Hey Bob, Want to go downtown to McDonald’s for lunch at 12:00AM?

11:38 AM. No response. Did Bob see the acknowledgement?

Heading to Dominos now. See you there!
Asynchronous Consensus: Texting

How could Alice and Bob find consensus on a time and place to meet for lunch?

Synchronous Consensus: Telephone

10:45 AM. Alice tries to invite Bob for lunch ...

Hey Bob, Want to go downtown to McDonald's for lunch at 12:00AM? 

How about a completo at Domino's instead?

Okay, 12:00AM?

See you then!

10:46 AM. Clear consensus!

Synchronous Consensus

• Can be guaranteed!
  – But only under certain conditions ...

What is the core difference between reaching consensus in synchronous (texting or email) vs. asynchronous (phone call) scenarios?

Synchronous Consensus: Telephone

10:45 AM. Alice tries to invite Bob for lunch ...

Hey Bob, Want to go downtown to McDonald's for lunch at 12:00AM?

How about a completo at Domino's instead?

Hello?

beep, beep, beep

10:46 AM. What's the protocol?

CAP Systems (for example ...)

CA: They are guaranteed to go to the same place for lunch as long as each of them can be reached.

CP: If someone cannot be reached, they all go to the same place for lunch or nobody meets.

AP: If someone cannot be reached, they all go downtown but might not end up at the same place.

(No intersection)

A Consensus Protocol

• Agreement/Consistency [Safety]: All working nodes agree on the same value. Anything agreed is final!
• Validity/Integrity [Safety]: Every working node decides at most one value. That value has been proposed by a working node.
• Termination [Liveness]: All working nodes eventually decide (after finite steps).
• Safety: Nothing bad ever happens
• Liveness: Something good eventually happens
A Consensus Protocol for Lunch

- **Agreement/Consistency [Safety]**: Everyone agrees on the same place downtown for lunch, or agrees not to go downtown.
- **Validity/Integrity [Safety]**: Agreement involves a place someone actually wants to go.
- **Termination [Liveness]**: A decision will eventually be reached (hopefully before lunch).

**Fault Tolerance:**

- **Fail–Stop Fault**
  - A machine fails to respond or times-out (often hardware or load)
  - Need at least $f+1$ replicated machines? (beware async.)
  
  \[ f = \text{number of clean failures} \]

- **Byzantine Fault**
  - A machine responds incorrectly/maliciously (often software)
  - Need at least $2f+1$ replicated machines?
  
  \[ f = \text{number of (possibly Byzantine) failures} \]

**Fail–Stop/Byzantine**

- Naively:
  - Need $f+1$ replicated machines for fail–stop
  - Need $2f+1$ replicated machines for Byzantine

- Not so simple if nodes must agree beforehand!

- Replicas must have consensus to be useful!

**Consenus Protocol:**

**Two-Phase Commit**
Two-Phase Commit (2PC)

- Coordinator & cohort members

- **Goal:** Either all cohorts commit to the same value or no cohort commits to anything

- Assumes synchronous, fail-stop behaviour
  – Crashes are known!

Two-Phase Commit (2PC)

**1. Voting:**
- I propose McDonalds! Is that okay?
  - Yes!
  - Yes!

Two-Phase Commit (2PC)

**2. Commit:**
- I have two yeses! Please commit.
  - Committed!
  - Committed!

Two-Phase Commit (2PC)

**1. Voting:**
- I propose McDonalds! Is that okay?
  - Yes!
  - No!

Two-Phase Commit (2PC) [Abort]

**2. Commit:**
- I don’t have two yeses! Please abort.
  - Aborted!
  - Aborted!

Two-Phase Commit (2PC) [Abort]

**1. Voting:**
- I propose McDonalds! Is that okay?
  - Yes!
  - Yes!

Two-Phase Commit (2PC) [Abort]

**2. Commit:**
- I don’t have two yeses! Please abort.
  - Aborted!
  - Aborted!

**Notes:**
- For *n* nodes, in the order of $4n$ messages.
  - $2n$ messages to propose value and receive votes
  - $2n$ messages to request commit and receive acks
Two-Phase Commit (2PC)

What happens if the coordinator fails?
• Cohort members know coordinator has failed!

Two-Phase Commit (2PC)

What happens if a coordinator and a cohort fail?
Not fault-tolerant!

Two-Phase Commit (2PC)

What happens if there’s a partition?
Not fault-tolerant!

CONSENSUS PROTOCOL:
THREE-PHASE COMMIT

Three-Phase Commit (3PC)

1. Voting:

Three-Phase Commit (3PC)

2. Prepare:
Three-Phase Commit (3PC)

3. Commit:

- Everyone is prepared. Please commit.

Committed!

Committed!

Three-Phase Commit (3PC)

1. Voting: (As before for 2PC)
2. Prepare: If all votes agree, coordinator sends and receives acknowledgements for a “prepare to commit” message
3. Commit: If all acknowledgements are received, coordinator sends “commit” message
   - For $n$ nodes, in the order of $6n$ messages.
     - $4n$ messages as for 2PC
     - $+2n$ messages for “prepare to commit”+ “ack.”

Three-Phase Commit (3PC)

What happens if the coordinator fails?

- Is everyone else prepared to commit?
- Yes!
- Prepared to commit!
- Okay! Committing!

Three-Phase Commit (3PC)

What happens if coordinator and a cohort member fail?

- Rest of cohort know if abort/commit!
- It’s a commit!

Three-Phase Commit (3PC)

Two-Phase vs. Three Phase

- Did you spot the difference?
- In 2PC, in case of failure, one cohort may already have committed/aborted while another cohort doesn’t even know if the decision is commit or abort!
- In 3PC, this is not the case!

Two/Three Phase Commits

- Assumes synchronous(-like) behaviour!
- Assumes knowledge of failures!
  - Cannot be guaranteed if there’s a network partition!
- Assumes fail–stop errors
How to decide the leader?

We need a leader for consensus ... so what if we need consensus for a leader?

CONSENSUS PROTOCOL: PAXOS

Turing Award: Leslie Lamport

- One of his contributions: PAXOS

LESLIE LAMPORT

- For fundamental contributions to the theory and practice of distributed and concurrent systems, notably the invention of concepts such as causality and logical clocks, safety and liveness, replicated state machines, and sequential consistency.

PAXOS Phase 1a: Prepare

- A coordinator proposes with a number \( n \)

PAXOS Phase 1b: Promise

- By saying "okay", a cohort agrees to reject lower numbers

PAXOS Phase 1a/b: Prepare/Promise

- This continues until a majority agree and a leader for the round is chosen ...
PAXOS Phase 2a: Accept Request

- The leader must now propose the value to be voted on this round ...

![Diagram showing the leader proposing a value and nodes voting on it.]

PAXOS Phase 2b: Accepted

- Nodes will accept if they haven’t seen a higher request and acknowledge ...

![Diagram showing nodes accepting a value and acknowledging it.]

PAXOS Phase 3: Commit

- If a majority pass the proposal, the leader tells the cohort members to commit ...

![Diagram showing the leader committing a value and members acknowledging it.]

PAXOS Summary

- Wait for majority
- Leader proposes
- Ignore "v" with lower ID
- Okay "v" sounds good
- We’re agreed on "v"

PAXOS: No Agreement?

- If a majority cannot be reached, a new proposal is made with a higher number (by another member)

PAXOS: Failure Handling

- Leader is fluid: based on highest ID the members have stored
  - If Leader were fixed, PAXOS would be like 2PC
- Leader fails?
  - Another leader proposes with higher ID
- Leader fails and recovers (asynchronous)?
  - Old leader superseded by new higher ID
- Partition?
  - Requires majority / when partition is lifted, members must agree on higher ID
PAXOS: Guarantees

- **Validity/Integrity:**
  - Value proposed by a leader

- **Agreement/Consistency:**
  - A value needs a majority to pass
  - Each member can only choose one value
  - Other proposals would have to try convince a majority node!
  - Therefore only one agreed value can be chosen!

PAXOS In-Use

Chubby: “Paxos Made Simple”

---

**LAB II REVIEW:**
EXTERNAL SORTING

What are (1) the strengths and (2) weaknesses of doing the word count (or other large-scale processing tasks) using external sorts compared with using main memory?

---

**LAB III PREVIEW:**
JAVA RMI OVERVIEW

Why is Java RMI Important?

We can use it to quickly build distributed systems using some standard Java skills.
What is Java RMI?

- RMI = Remote Method Invocation
- Remote Procedure Call (RPC) for Java
- Ancestor of CORBA (in Java)
- Stub / Skeleton model (TCP/IP)

Stub/Skeleton Same Interface!

Server Implements Skeleton

Server Registry

Server Creates/Connects to Registry
Server Registers Skeleton Implementation As a Stub

```java
"create a remote stub to use it
// needs for incoming calls
Remote stub = new JNDIRemoteObject("testReg:Stub");
// register stub in registry under a key stub-name
String stubName = "sk1";
registry.add(stubName, stub);
```
RECAP

CAP Systems

CA: Guarantees to give a correct response but only while network works fine (Centralised / Traditional)

CP: Guarantees responses are correct even if there are network failures, but response may fail (Weak availability)

AP: Always provides a "best-effort" response even in presence of network failures (Eventual consistency)

(No intersection)

Consensus for CP-systems

• Synchronous vs. Asynchronous
  – Synchronous less difficult than asynchronous

• Fail–stop vs. Byzantine
  – Byzantine typically software (arbitrary response)
  – Fail–stop gives no response

Consensus for CP-systems

• Two-Phase Commit (2PC)
  – Voting
  – Commit

• Three-Phase Commit (3PC)
  – Voting
  – Prepare
  – Commit

Consensus for CP-systems

• PAXOS:
  – 1a. Prepare
  – 1b. Promise
  – 2a. Accept Request
  – 2b. Accepted
  – 3. Commit

Java: Remote Method Invocation

• Java RMI:
  – Remote Method Invocation
  – Stub on Client Side
  – Skeleton on Server Side
  – Registry maps names to skeletons/servers
  – Server registers skeleton with key
  – Client finds skeleton with key, casts to stub
  – Client calls method on stub
  – Server runs method and serialises result to client
Questions?