Coordination

Professor Natacha Crooks
https://cs162.org/

Slides based on prior slide decks from David Culler, Ion Stoica, John Kubiatowicz, Alison Norman and Lorenzo Alvisi
Recall: End To End Principle

Think twice before implementing functionality in the network

If hosts can implement functionality correctly, implement it in a lower layer only as a performance enhancement

But do so only if it does not impose burden on applications that do not require that functionality

This is the interpretation we are using
Recall: Map Reduce

How can we implement word count using only map and reduce?

Three steps:

1) convert files into pairs of (key, value)
2) Define a map function. Apply to all files
3) Shuffle! All elements with same key go to same reduce
4) Define a reduce function. Apply to result of the map function.
Recall: Map Reduce

0.txt
hello world

1.txt
hello edward

2.txt
hello rahul

\[ \begin{align*}
(0.txt, "hello world") & \xrightarrow{\text{map}} \{("hello", 1), ("world", 1)\} \\
(1.txt, "hello edward") & \xrightarrow{\text{map}} \{("hello", 1), ("edward", 1)\} \\
(2.txt, "hello rahul") & \xrightarrow{\text{map}} \{("hello", 1), ("rahul", 1), ("hello", 1)\}
\end{align*} \]

Output.txt

hello, 4
edward, 1
rahul, 1
Recall: Idempotence is back!

When a worker fails, simply retry failed tasks!

Since failed tasks are retried, application map and reduce functions generally should be pure, deterministic functions of their arguments.

Should not depend on the current time, randomness, resources accessed over the network, etc.

Tasks that are not pure functions can be run on MapReduce, but the results may or may not be cohesive.
Topic roadmap

- Distributed File Systems
- Peer-To-Peer System: The Internet
- Distributed Data Processing
- Coordination (Atomic Commit and Consensus)
Coordination: making distributed decisions

Functionality is spread across machines. Requires coordination to reach distributed decision

Distributed Protocols are hard!
Coordination is hard!

When machines can fail!

When networks are slow and/or unreliable

When machines may receive conflicting proposals on what to do
Coordination: making distributed decisions

Accept if all machines accept

Accept?
YES

Accept?
YES

Accept?
YES

Client: persist my data

Flush to disk

Flush to disk

Flush to disk

Your data is persisted!
Agreeing simultaneously: General’s Paradox

Two generals, on separate mountains
   »Can only communicate via messengers
   »Messengers can be captured

Problem: need to coordinate attack
   »If they attack at different times, they all die
   »If they attack at same time, they win
General’s Paradox: Scenario 1

Attack at 11 am!
General’s Paradox: Scenario 1

Is it safe for both of them to attack?
No! Caesar doesn’t know that Brutus received the message!
General’s Paradox: Scenario 1

Attack at 11 am!

Sends an ACK to Caesar!

Yes! Attack at 11 am!
**General’s Paradox: Scenario 1**

**Attack at 11 am!**

**Now is it safe?**

**No! Messenger could have been attacked**

**Yes! Attack at 11 am!**
General's Paradox: Scenario 1

Caesar needs to know that
Brutus knows that
Caesar knows that
Brutus knows that
They are attacking at 11 am

Impossible to achieve simultaneous actions with unreliable channels because we never know whether messenger or ACK got lost
Agreeing simultaneously: General’s Paradox

If the network is unreliable, it is impossible to guarantee two entities do something simultaneously.

If nodes behave maliciously, impossible to get eventual agreement if there are less than $3f+1$ parties present (of which $f$ can misbehave).

Entire textbook on impossibility results in distributed computing ...
**Eventual Agreement: Two-Phase Commit**

Two or more machines agree to do something, or not do it, **atomically**

No constraints on time, just that it will eventually happen!

*Used in most modern distributed systems! Representative of other coordination protocols*
Eventual Agreement: Two-Phase Commit

Developed by Turing award winner Jim Gray
–(first Berkeley CS PhD, 1969)

Many important Database breakthroughs also from Jim Gray

Jim Gray
Eventual Agreement: Two-Phase Commit

Goal: determine whether should commit or abort a transaction

All processes that reach a decision reach the same one (Agreement)

A process cannot reverse its decision after it has reached one (Finality)

If there are no failures and every process votes yes, the decision will be commit (Consistency)

If all failures are repaired and there are no more failures, then all processes will eventually decide commit/abort (Termination)


2PC Terminology

Setup:
- One coordinator
- A set of participants

Each process has access to a persistent log:

Processes can crash and recover.

Recorded information on the log will persist after crashes.
2PC Terminology

Coordinator asks all processes to vote

Each participant (including coordinator) can vote either YES or NO

- If all vote YES, coordinator must vote COMMIT
- If one of them votes NO, coordinator must vote ABORT
2PC: The easy case (No failures)

**Coordinator Algorithm**

1. Coordinator sends VOTE-REQ to all workers

3. Collect votes
   - If receive VOTE-COMMIT from all N workers, send GLOBAL-COMMIT to all workers
   - If don’t receive VOTE-COMMIT from all N workers, send GLOBAL-ABORT to all workers

**Worker Algorithm**

2. – Send VOTE-COMMIT or VOTE-ABORT to coordinator
   – If sent VOTE-ABORT immediately abort

4. - If receive GLOBAL-COMMIT then commit
   – If receive GLOBAL-ABORT then abort
Failure Free Example Execution

coordinator

worker 1

worker 2

worker 3

time

VOTE-REQ

GLOBAL-COMMIT

VOTE-COMMIT

VOTE-COMMIT
Coordinator implements simple state machine

Coordinator implements simple state machine

State Machine of Coordinator

INIT

WAIT

ABORT

COMMIT

Recv: START
Send: VOTE-REQ

Recv: VOTE-ABORT
Send: GLOBAL-ABORT

Recv: all VOTE-COMMIT
Send: GLOBAL-COMMIT
What about failures?

1. Coordinator sends VOTE-REQ to all workers

2. – Send VOTE-COMMIT or VOTE-ABORT to coordinator
   – If sent VOTE-ABORT immediately abort

3. Collect votes
   – If receive VOTE-COMMIT from all N workers, send GLOBAL-COMMIT to all workers
   – If don’t receive VOTE-COMMIT from all N workers, send GLOBAL-ABORT to all workers

4. - If receive GLOBAL-COMMIT then commit
   – If receive GLOBAL-ABORT then abort

1) What happens when waiting for a message that never comes?

2) What happens during when participant recovers from a failure?
**What happens when a message never comes?**

**Coordinator Algorithm**

1. Coordinator sends **VOTE-REQ** to all workers

2. Send **VOTE-COMMIT** or **VOTE-ABORT** to coordinator
   - If sent **VOTE-ABORT**, immediately abort

3. Collect votes
   - If receive **VOTE-COMMIT** from all N workers, send **GLOBAL-COMMIT** to all workers
   - If don’t receive **VOTE-COMMIT** from all N workers, send **GLOBAL-ABORT** to all workers

4. If receive **GLOBAL-COMMIT**
   - Then commit
   - If receive **GLOBAL-ABORT**
     - Then abort

**Worker Algorithm**

- Step 2: worker waiting from **VOTE-REQ** from coordinator
- Step 3: Coordinator is waiting for vote from participants
- Step 4: Worker who voted YES is waiting for decision
What happens when a message never comes?

**Coordinator Algorithm**

1. Coordinator sends **VOTE-REQ** to all workers

**Worker Algorithm**

2. 
   - Send **VOTE-COMMIT** or **VOTE-ABORT** to coordinator
   - If sent **VOTE-ABORT** immediately abort

3. Collect votes
   - If receive **VOTE-COMMIT** from all N workers, send **GLOBAL-COMMIT** to all workers
   - If don’t receive **VOTE-COMMIT** from all N workers, send **GLOBAL-ABORT** to all workers

4. 
   - If receive **GLOBAL-COMMIT** then commit
   - If receive **GLOBAL-ABORT** then abort

**Step 2:** worker waiting from **VOTE-REQ** from coordinator

*Since it has not cast its vote yet, worker can decide abort and halt*

**Step 3:** Coordinator is waiting for vote from participants

*Coordinator can always vote abort herself, so votes abort and sends **GLOBAL-ABORT** to all participants*

**Step 4:** Worker who voted **COMMIT** is waiting for decision

*Worker cannot decide: it must run a termination protocol*
Termination Protocol

• Option 1: Simply wait for coordinator to recover.

If all failures are repaired and there are no more failures, then all processes will eventually decide commit/abort (Termination)

=> No need to recover until coordinator has recovered

• (Better) Option 2: Ask a friendly participant p

Case 1: If p has decided COMMIT/ABORT, forwards decision to initiator

Case 2: If P has not decided, votes ABORT, sends abort to initiator. Initiator knows decision will be ABORT. So can decide

Case 3: If P has voted COMMIT, P is also stuck and can’t help initiator

If every participant voted COMMIT and coordinator crashes before sending decision, must wait for coordinator to recover to decide!
Example of Coordinator Failure #1

coordinator

worker 1

worker 2

worker 3

VOTE-REQ

timeout

timeout

timeout

VOTE-ABORT
Example of Coordinator Failure #2

coordinator

VOTE-REQ

VOTE-COMMIT

worker 1

block waiting for coordinator

worker 2

worker 3

restarted

GLOBAL-ABORT
Machine recovery

All nodes use **stable storage** to store current state (e.g. backed by disk/SSD).
Upon recovery, nodes can restore state and resume

When coordinator sends **VOTE-REQ**, writes **START-2PC** to log

=> **Coordinator reads log**, if sees **VOTE-REQ** but no decision, decides **ABORT** unilaterally
Before voting, participant writes **VOTE-*** to stable log, then sends vote

=> **Participant reads log**, if doesn’t see record, sends **VOTE-ABORT**. If **VOTE-COMMIT**, contacts friend
Before sending decision, coordinator writes **GLOBAL-*** to stable log, then sends decision

=> **Coordinator reads log**, if sees **GLOBAL-***, resends decision

After receiving **GLOBAL-***, participant writes commit/abort to stable log

=> **Participants read log**, 2PC instance has already been terminated
2PC Summary

Why is 2PC not subject to the General’s paradox?
- Because 2PC is about all nodes eventually coming to the same decision - not necessarily at the same time!
- Allowing us to reboot and continue allows time for collecting and collating decisions

Biggest downside of 2PC: blocking
- A failed node can prevent the system from making progress
- Still one of the most popular coordination algorithms today
**Alternatives to 2PC**

**Three-Phase Commit:** One more phase, allows nodes to fail or block and still make progress.

**PAXOS:** An alternative used by Google and others that does not have 2PC blocking problem
- Developed by Leslie Lamport (Turing Award Winner)
- No fixed leader, can choose new leader on fly, deal with failure

What happens if one or more of the nodes is malicious?
- **Malicious:** attempting to compromise the decision making
- Use a more hardened decision-making process: Byzantine Agreement and Blockchains