AGREEMENT PROTOCOLS

Paxos - a family of protocols for solving consensus
OUTLINE

- History of the Paxos algorithm
- Paxos Algorithm Family
- Implementation in existing systems
- References
HISTORY OF THE PAXOS ALGORITHM

- Discovered by Lamport in the late 80s
  - ...while trying to prove that a complicated algorithm for handling Byzantine failures used in a program was impossible.

- Contains the first 3-phase-commit algorithm [2] which had
  - a clearly stated correctness condition
  - a proof of correctness

- He used humor to describe the algorithm
  - Created a lost civilization and described the decision-making in the parliament of the Greek island of Paxos
  - Used names of computer scientists for the Greek legislators, bogus Greek dialect
  - Gave some lectures dressed as an Indiana Jones-like character
  - This was, by his own words, a failure, as the readers didn’t get the importance of the algorithm!
  - After some years, when interest grew, he republished the article including a funny annotation from Keith Marzullo.
THE PART-TIME PARLIAMENT [1/2]

- Short overview of the part-time parliament
  - Legislators (=processes)
    - determined the laws
  - Messengers (=messages)
    - delivered messages between the legislators
  - Location
    - Chamber of parliament
    - Legislators/Messengers could leave the chamber (failing process/message)

- Problems
  - No legislator was willing to remain through a whole session to act as secretary and record the actions

  - The messenger was trusted that when he delivered, he delivered a correct message, but the messenger could suddenly leave the chamber for a short time, or take a 3-week holiday or simply never return

  - The messenger had a bad memory of whether he had delivered the message or not and could deliver it several times.
THE PART-TIME PARLIAMENT [2/2]

- **Solution**
  - Each legislator maintained a ledger with inedible ink which contained the numbered sequence of the decrees that were passed
    - The back of the ledger was eraseable and used for notes

- **Requirements**
  - *Consistency* of ledgers - no two ledgers can be inconsistent
    - If one had the entry "132: Lamps must only use olive oil", not other ledger could have another entry for decree 132.
  - To avoid only blank ledgers, they had a *progress condition*
    - As long as the majority of the legislators were in Chamber and they didn’t leave or enter in a long enough period of time, any proposed law would be passed and be noted on the ledgers.
Paxos Algorithms Family

- Basic Paxos
- Multi-Paxos
- Cheap Paxos
- Fast Paxos
- Generalized Paxos
- Byzantine Paxos
**Consenus**

- Agreeing on one result among a group of participants
- Consensus protocols are the basis for the *state machine approach* in distributed computing
- Difficult to achieve when the participants or the network medium experience faults
FAULT-TOLERANCE

○ The Paxos family of algorithm’s main property is safety from inconsistency

○ Safety requirement:
  • Nontriviality
    ○ only *proposed* values are learned
  • Consistency
    ○ *one* value at most can be learned

○ Liveness requirement:
  • if a value $v$ has been learned, then some learner will eventually learn some value
Preliminaries

- **Processes:**
  - Experience failures, may rejoin, *do not collude, lie or divert the protocol* (Byzantine Paxos)

- **Network:**
  - Processes can send messages to any other process
  - Messages are sent asynchronously, may be lost, duplicated, reordered, *delivered without corruption*

- *When the protocol is run in 2F+1 processes, F processes can fail. When applying reconfiguration, more than F processes may fail, if failures do not occur too soon after each other*
Basic Paxos

- **Roles:**
  - **Client:** issues a *request*, waits for *response*
  - **Acceptor:** memory of the protocols, collected into *Quorums*
  - **Proposer:** proposes/advocates the Client’s request, convinces the Acceptors, resolves conflicts
  - **Learner:** acts as the *replication* factor, takes action after Acceptors agree on Client’s request
  - **Leader:** a distinguished Proposer, necessary to make progress
  - **Quorum:** any majority of participating Acceptors
  - **Choice:** Leader chooses a value from most recent round if a set of values is in conflict
Basic Paxos is **not** the protocol typically implemented in a deployment.

Each instance decides only on a **single** value and has two phases:

**Phase 1:**
- **A) Prepare:** Leader selects a proposal number $N$ and sends a **prepare message** to the quorum of Acceptors.

- **B) Promise:** if $N$ larger than previous proposal, Acceptors send a **promise message** not to accept proposals less than $N$, otherwise a denial, (Nack), is sent back.
BASIC PAXOS (CONTD.)

- **Phase 2:**
  - **A) Accept!**: if Leader receives response from Quorum of Acceptors, it may choose a value to be agreed upon, sends an *accept message* to Quorum of Acceptors with chosen value.
  - **B) Accepted**: when Acceptors receive an accept message for a proposal they have promised, they accept, and send an *accepted message* to the Proposer and every Learner.
Basic Paxos (contd.)

One instance, one successful round

```
Client  Proposer  Acceptor  Learner
|       | | | | | Request
|       | | | | | Prepare(N)
|       | | | | | Promise(N,[V_a, V_b, V_c])
|       | | | | | Accept!(N,V_n)
|       | | | | | Accepted(N,V_n)
|<---------------------X----X----------------| Response
|       | | | | |
```
**Basic Paxos - Failures**

<table>
<thead>
<tr>
<th>Client</th>
<th>Proposer</th>
<th>Acceptor</th>
<th>Learner</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X--------&gt;</td>
<td></td>
<td>Request</td>
</tr>
<tr>
<td></td>
<td>X---------&gt;</td>
<td>-&gt;</td>
<td>-&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>!</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;-------X--X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X---------&gt;</td>
<td>-&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;-------X--X-------&gt;</td>
<td>-&gt;</td>
</tr>
<tr>
<td>&lt;--------------------------</td>
<td>--X</td>
<td>Response</td>
<td></td>
</tr>
</tbody>
</table>

- Failure of one Acceptor
- Quorum of Acceptors remains alive
- No recovery, no additional rounds or messages
BASIC PAXOS – FAILURES (contd.)

- Failure of redundant Learner
- No recovery, no additional rounds or messages
BASIC PAXOS – FAILURES (contd.)

- Proposer fails after proposing a value
- New Leader is elected, one extra round
<table>
<thead>
<tr>
<th>Client</th>
<th>Proposer</th>
<th>Acceptor</th>
<th>Learner</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
<tr>
<td>( x \leftarrow x \rightarrow</td>
<td>)</td>
<td>( x \rightarrow</td>
<td>)</td>
</tr>
</tbody>
</table>

- Multiple Proposers believe to be Leaders
- Most complex case
- Safety properties are preserved regardless

... and so on ...
**Multi-Paxos**

- Basic Paxos continuesly repeated
- Most common deployment of the Paxos family
- The **distributed state machine** gets it’s commands from a continuous stream of agreed values
- *If the Leader is stable, Phase 1 can be skipped for future instances*

<table>
<thead>
<tr>
<th>Client</th>
<th>Proposer</th>
<th>Acceptor</th>
<th>Learner</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X--------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Request</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Accept!</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Accepted</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Response</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

--- Following Requests ---
OPTIMIZATION POSSIBILITIES

1) Single distinguished learner informs the other learners. Acceptors sends ”Accepted” only to distinguished learner. Saves 1 message delay.

2) Leader send Prepare/Accept! only to a quorom of Acceptors. *(This is the key for CHEAP PAXOS)*

3) If an Acceptor learns the chosen value, it can store it. When receiving phase 1a/2a message, it can skip phase 1b/2b and inform the Leader about the chosen value

4) Use hash value

5) *Phase 1 is unnecassary for the first round. Leader of round 1 can start by sending an Accept! Message with any proposed value*
Cheap Paxos (1)

- Extends Basic Paxos

- Tolerates $F$ failures using
  - $F+1$ main processes
  - $F$ auxillary processes
    - Performs actions only if a main process fails

- Reconfigures the system if a main process fails
**Cheap Paxos (2)**

<table>
<thead>
<tr>
<th>Proposer</th>
<th>Acceptor</th>
<th>Main</th>
<th>Aux</th>
<th>Learner</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image.png" alt="Diagram" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Failure scenario: Main process fails**
  - If a main process fails, another quorum containing one or more auxiliary process(es) is used to complete the execution.
  
  - Reconfigures the system so that the set of all working processes forms a quorum.
Cheap Paxos (3)

- **Pros**
  - Reduces processor requirements
    - Processors used for Auxillary processes do not need as much processing power or storage as the main processors

- **Cons**
  - Reduces liveness property
    - If too many processes fail, the system will halt until auxillary processes have reconfigured the system
    - With the assumption that the processes do not move around too much, then liveness is maintained
FAST PAXOS

- Extension of Basic Paxos, reduces end-to-end message delays
  - Basic Paxos
    - 3 message delays from request to learning
  - Fast Paxos
    - 2 message delays from request to learning, assuming no collisions.
    - 3 with collision.
**FAST PAXOS - NON-CONFLICTING FLOW**

- If the leader has no value to propose, it can send an *Any* message to the *Acceptors*
  - The *Acceptor* will then treat any received proposing message as a *Accept!* (phase 2a) message with the given value.

- *Does not guarantee consistency*
  - Each *Acceptor* decides independently which proposal message to take as a phase 2a message and therefore they may vote for different values.
  - This may cause a collision.
### Fast Paxos - Conflicting Proposals

<table>
<thead>
<tr>
<th>Client</th>
<th>Leader</th>
<th>Acceptor</th>
<th>Learner</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Diagram" /></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Coordinated recovery
- If the *Leader* detects a collision when receiving the different *phase 2b* messages, it solves this by:
  - treating them as *phase 1b* messages
  - then executes a round of *phase 2a messages* to the Acceptors.
  - 4 message delays from request to learning

#### Uncoordinated recovery
- The *Leader* can specify a recovery technique in advance
  - Allows the *Acceptors* to recover from the collision themselves
  - 3 message delays from request to learning
FAST PAXOS - CONFLICTING PROPOSALS

- Coordinated/Uncoordinated recovery add new actions to the protocol
  - Maintains safety
FAST PAXOS - **COLLAPSED ROLES**

- Acceptor/Learner is merged
  - Only 2 message delays from request to learning
GENERALIZED PAXOS (1)

- Extending the algorithm from agreeing on one value, to agreeing on an increasing set of values
- Commands commute
  - If executing them in any order will give the same result
GENERALIZED PAXOS (2)

- **Requirements**
  - **Nontriviality**
    - For any learner $l$, the value of $learned[l]$ is always a sequence of proposed commands.
  - **Stability**
    - For any learner $l$, the value of $learned[l]$ at any time is a prefix of its value at a later time.
  - **Consistency**
    - For learners $l_1$, $l_2$, the sequences $learned[l_1]$ and $learned[l_2]$ is a prefix of the other.
  - **Liveness**
    - If a command $C$ has been proposed, then eventually the sequence $learned[l]$ will contain the element $C$.
GENERALIZED PAXOS (3)

- Two concurrent clients performing operations on two register addresses, A and B
- X indicates which concurrent operations may cause an interference
- A proposed series of operations
- A possible permutation allowed by commutes
### Generalized Paxos (4)

<table>
<thead>
<tr>
<th>Client</th>
<th>Leader</th>
<th>Acceptor</th>
<th>Learner</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
// New Leader Begins Round
Prevent(N)
Promise(N, null)
Phase2Start(N, null)

// Concurrent comming proposals
Propose(ReadA)
Propose(ReadB)
Accepted(N, <ReadA, ReadB>)
Accepted(N, <ReadB, ReadA>)

// No Conflict, both accepted
Stable - <ReadA, ReadB>

// Concurrent conflicting proposals
Propose(ReadB)
Accepted(N, <WriteB, ReadA>, <ReadB>)
Accepted(N, <ReadB>, <WriteB, ReadA>)

// Conflict detected, leader chooses commutative order:
// V = <ReadA, WriteB, ReadB>
Phase2Start(N1, V)
Accepted(N1, V)
Stable - <ReadA, ReadB>, <ReadA, WriteB, ReadB>

// More conflicting proposals
Propose(WriteA)
Propose(ReadA)

// Leader chooses order W
Phase2Start(N2, W)
Accepted(N2, W)
Stable - <ReadA, ReadB>, <ReadA, WriteB, ReadB>, <WriteA, ReadA>
```
**Byzantine Paxos**

- Supports arbitrary failures of participants
- Adds an extra *Verify* message
  - verifies actions of the other participants
  - distributes knowledge
Fast Byzantine Paxos

- Clients send messages directly to Acceptors.
- Accepted message is sent to all Acceptors and Learners.
- Since the Verify message is removed, it has 1 message delay less than Byzantine Paxos.
BYZANTINE PAXOS

- Failure scenario for both protocols
  - Each Learner waits until it receives $F+1$ identical messages from different Acceptors
  - If this doesn’t happen, the Acceptors will also know about the failure
  - The Acceptors without failure will then re-broadcast the agreed value

<table>
<thead>
<tr>
<th>Client</th>
<th>Acceptor</th>
<th>Learner</th>
</tr>
</thead>
</table>
| ![Diagram](image.png) | | !! One Acceptor is faulty
| ![Diagram](image.png) | | Accept!(N,I,V)
| ![Diagram](image.png) | | !! Learners receive 2 different commands
| ![Diagram](image.png) | | Accepted(N,I,{V,W}) - BROADCAST
| ![Diagram](image.png) | | !! Correct Acceptors notice error and choose
| ![Diagram](image.png) | | Accepted(N,I,V) - BROADCAST
| ![Diagram](image.png) | | Response(V)
GOOGLE CHUBBY

- A fault-tolerant system at Google, provides a distributed locking mechanism and stores small files
- Achieves fault-tolerance through replication
- Uses Paxos as the base for a framework that implements a fault-tolerant log
- Clients contact a Chubby cell, the Master replica serves all client requests
- A Chubby cell consists of usually 5 replicas
- If the Master fails, new Master is automatically elected
Google Chubby

Chubby clients network

Replicas network

Snapshot exchange

Paxos protocol

Chubby protocol

Chubby

Fault-tolerant DB

Fault-tolerant Log

Local file system

File I/O

Log

Snapshot
Other systems using the Paxos algorithm

- Yahoo ZooKeeper [6,7]
  - a “high available and reliable coordination system”
  - is used by distributed applications
  - uses a version of the Paxos algorithm to keep replicas consistent.

- Microsofts Autopilot [8]
  - an automatic data center management infrastructure
  - the largest deployed cluster managed by a single instance of Autopilot contains up to tens of thousands computers
  - All info about the state the system should be in is held in a state machine called Device Manager
    - DM is distributed over 1-10 computers
  - Paxos is used to keep consensus between replicas, batching updates to get a balance between latency and throughput
REFERENCES

7. Hoff, T. "ZooKeeper – A Reliable, Scalable Distributed Coordination System", 2008 URL: http://highscalability.com/node/426