Outline

1. Consensus
2. The Part-Time Parliament
3. Single-Decree Paxos
4. Liveness
5. Multi-Decree Paxos
6. Paxos Variants
7. Conclusion
Outline

1. **Consensus**
2. The Part-Time Parliament
3. Single-Decree Paxos
4. Liveness
5. Multi-Decree Paxos
6. Paxos Variants
7. Conclusion
What is consensus?

Where do we want to go to eat lunch?
What is consensus?

I personally don’t care. Indifferent.

I’m good with anywhere.
What is consensus?

Where do we want to go to eat lunch?
What is consensus?

I’d like Thai food.

I’m feeling Korean food.

I also want Thai food.
What is consensus?

OK, let’s get Thai food.

OK, let’s get Thai food.

OK, let’s get Thai food.
What is consensus?

*Consensus* is the problem of getting a set of processors to agree on some value.
What is consensus?

OK, let’s get Thai food.

OK, let’s get Thai food.

OK, let’s get Thai food.
What is consensus?

More formally, *consensus* is the problem of satisfying the following properties:

- Validity
- Agreement
- Integrity
- Termination
What is consensus?

More formally, consensus is the problem of satisfying the following properties:

- **Validity**
  - If all processes that propose a value propose v, then all correct deciding processes eventually decide v
- **Agreement**
- **Integrity**
- **Termination**
What is consensus?

Validity: If all processes that propose a value propose v, then all correct deciding processes eventually decide v
What is consensus?

More formally, *consensus* is the problem of satisfying the following properties:

- **Validity**
  - If all processes that propose a value propose $v$, then all correct deciding processes eventually decide $v$
- **Agreement**
  - If a correct deciding process decides $v$, then all correct deciding processes eventually decide $v$
- **Integrity**
- **Termination**
What is consensus?

Agreement: If a correct deciding process decides $v$, then all correct deciding processes eventually decide $v$.
What is consensus?

More formally, *consensus* is the problem of satisfying the following properties:

- **Validity**
  - If all processes that propose a value propose v, then all correct deciding processes eventually decide v

- **Agreement**
  - If a correct deciding process decides v, then all correct deciding processes eventually decide v

- **Integrity**
  - Every correct deciding process decides at most one value, and if it decides v, then some process must have proposed v

- **Termination**
What is consensus?

I’d like **Thai** food.

I’m feeling **Korean** food.

I also want **Thai** food.

**Integrity:** Every correct deciding process decides at most one value, and if it decides v, then some process must have proposed v.
What is consensus?

More formally, consensus is the problem of satisfying the following properties:

- **Validity**
  - If all processes that propose a value propose v, then all correct deciding processes eventually decide v

- **Agreement**
  - If a correct deciding process decides v, then all correct deciding processes eventually decide v

- **Integrity**
  - Every correct deciding process decides at most one value, and if it decides v, then some process must have proposed v

- **Termination**
  - Every correct learning process eventually learns some decided value
What is consensus?

OK, let’s get Thai food.

OK, let’s get Thai food.

OK, let’s get Thai food.

Termination: Every correct learning process eventually learns some decided value
Assumption about our model

- **Asynchronous**, but **reliable**, network
Assumption about our model

- **Asynchronous**, but **reliable** network
  - Every message is eventually delivered, but can be delayed arbitrarily long
Assumption about our model

- **Asynchronous, but reliable, network**
  - Every message is eventually delivered, but can be delayed arbitrarily long
  - Processes can take arbitrarily long to transition between states
Assumption about our model

- **Asynchronous, but reliable, network**
  - Every message is eventually delivered, but can be delayed arbitrarily long
  - Processes can take arbitrarily long to transition between states
- Processes can only fail by **crashing**
Assumption about our model

- **Asynchronous, but reliable, network**
  - Every message is eventually delivered, but can be delayed arbitrarily long
  - Processes can take arbitrarily long to transition between states

- **Processes can only fail by **crashing**
  - No indication of failure; simply stops responding to messages
Assumption about our model

- **Asynchronous, but reliable, network**
  - Every message is eventually delivered, but can be delayed arbitrarily long
  - Processes can take arbitrarily long to transition between states

- **Processes can only fail by crashing**
  - No indication of failure; simply stops responding to messages
  - Failed processes cannot arbitrarily transition or send arbitrary messages
Timeline

Time, Clocks and Ordering

State Machine Replication

Paxos Published

1978  1984  1989
Timeline

- Time, Clocks and Ordering: 1978
- State Machine Replication: 1984
- Paxos Published: 1989
- Paxos Published In Journal: 1998
Timeline

Time, Clocks and Ordering

State Machine Replication

Paxos Published

Paxos Published In Journal

Paxos Made Simple

Timeline

Time, Clocks and Ordering
1978

State Machine Replication
1984

Paxos Published
1989

Paxos Published In Journal
1998

Paxos Made Simple
2001

Paxos Made Moderately Complex
2015
Recall the Consensus Problem in the State Machine Approach
Outline

1. Consensus
2. The Part-Time Parliament
3. Single-Decree Paxos
4. Liveness
5. Multi-Decree Paxos
6. Paxos Variants
7. Conclusion
Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxon parliament’s protocol provides a new way of implementing the state machine approach to the design of distributed systems.
The Part-Time Parliament
The Part-Time Parliament


Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxon parliament’s protocol provides a new way of implementing the state machine approach to the design of distributed systems.

- Paxos Made Simple (2001)

The Paxos algorithm, when presented in plain English, is very simple.
This article explains the full reconfigurable multidecree Paxos (or multi-Paxos) protocol. Paxos is by no means a simple protocol, even though it is based on relatively simple invariants. We provide pseudocode and explain it guided by invariants.
Outline

1. Consensus
2. The Part-Time Parliament
3. **Single-Decree Paxos**
4. Liveness
5. Multi-Decree Paxos
6. Paxos Variants
7. Conclusion
Roles in Protocol

- **Validity**
  - If all processes that propose a value propose $v$, then all correct deciding processes eventually decide $v$

- **Agreement**
  - If a correct deciding process decides $v$, then all correct deciding processes eventually decide $v$

- **Integrity**
  - Every correct deciding process decides at most one value, and if it decides $v$, then some process must have proposed $v$

- **Termination**
  - Every correct learning process eventually learns some decided value
Roles in Protocol

- **Validity**
  - If all processes that propose a value propose v, then all correct deciding processes eventually decide v

- **Agreement**
  - If a correct deciding process decides v, then all correct deciding processes eventually decide v

- **Integrity**
  - Every correct deciding process decides at most one value, and if it decides v, then some process must have proposed v

- **Termination**
  - Every correct learning process eventually learns some decided value
Constructing a Protocol

Proposer

Do nothing

Acceptor

Let $v_{\text{decided}} = v_0$ and send $\text{decide}(v_0)$ to learners
Constructing a Protocol

Integrity: Every correct deciding process decides at most one value, and if it decides $v$, then some process must have proposed $v$.
Constructing a Protocol

Proposer

When have value \( v \) to propose

- Send \( \text{propose}(v) \) to acceptors

Acceptor

On receive \( \text{propose}(v) \)

- If not yet decided, let \( v_{\text{decided}} = v \) and send \( \text{decide}(v) \) to learners
Constructing a Protocol

Termination: Every correct learning process eventually learns some decided value
Constructing a Protocol

Proposer
When have value v to propose

- Send propose(v) to acceptors

Acceptor
On receive propose(v)

- If not yet decided, let $v_{\text{decided}} = v$

When majority of correct acceptors have decided v

- Send decide(v) to learners
Constructing a Protocol

propose(v) → decide(v)
propose(v) → decide(v)
propose(v) → decide(v)
propose(v) → decide(v)
Constructing a Protocol

**Agreement**: If a correct deciding process decides \( v \), then all correct deciding processes eventually decide \( v \)
Constructing a Protocol

Ballot number: unique natural number associated with each proposal made by any proposer
Constructing a Protocol

Proposer

When have value v to propose

- Send prepare(b) to acceptors, where b is the highest ballot number not yet used that is known to the proposer

When have majority of acceptors’ promises for proposal b

- Send propose(v,b) to acceptors

Accepter

On receive prepare(b)

- If b > b_{promised}, let b_{promised} = b and respond with promise(b)

On receive propose(v,b)

- If b = b_{promised}, let v_{decided} = v

When majority of correct acceptors have decided v

- Send decide(v) to learners
Constructing a Protocol

**Integrity:** Every correct deciding process decides at most one value, and if it decides \( v \), then some process must have proposed \( v \)
Constructing a Protocol

**Proposer**

When have value v to propose
- Send prepare(b) to acceptors, where b is the highest ballot number not yet used that is known to the proposer

When have majority of acceptors’ promises for proposal b
- Send propose(v,b) to acceptors, where v is the value of the highest accepted proposal, or any value if no proposal accepted

**Acceptor**

On receive prepare(b)
- If b > \(b_{\text{promised}}\), let \(b_{\text{promised}} = b\) and respond with \(\text{promise}(b, v_{\text{decided}})\)

On receive propose(v,b)
- If b = \(b_{\text{promised}}\), let \(v_{\text{decided}} = v\)

When majority of correct acceptors have decided v
- Send decide(v) to learners
Constructing a Protocol Paxos

Proposer

When have value v to propose

- Send prepare(b) to acceptors, where b is the highest ballot number not yet used that is known to the proposer

When have majority of acceptors’ promises for proposal b

- Send propose(v,b) to acceptors, where v is the value of the highest accepted proposal, or any value if no proposal accepted

Acceptor

On receive prepare(b)

- If b > b\text{promised}, let b\text{promised} = b and respond with promise(b, v\text{decided})

On receive propose(v,b)

- If b = b\text{promised}, let v\text{decided} = v

When majority of correct acceptors have decided v

- Send decide(v) to learners
Outline

1. Consensus
2. The Part-Time Parliament
3. Single-Decree Paxos
4. Liveness
5. Multi-Decree Paxos
6. Paxos Variants
7. Conclusion
Liveness

- Something good eventually happens
  - Progress is made
    - An action is always eventually executed
Liveness

- Something good eventually happens
  - Progress is made
    - An action is always eventually executed
- In consensus
  - Termination
    - Every correct learning process eventually learns some decided value
Liveness

- Something good eventually happens
  - Progress is made
    - An action is always eventually executed
- In consensus

- Termination
  - Every correct learning process eventually learns some decided value

Does Paxos guarantee liveness?
Scenario

prepare(0)
Scenario
Scenario

prepare(1) → 0 → 0 → 0
Scenario

promise(1)
Scenario

propose(v, 0)
Scenario

prepare(3)

3

3

3

3
Scenario

promise(3) → 3 → 3 → 3 → 😞
Scenario

propose(v', 1)
Scenario

prepare(4) → 3 → 3 → 3 → 3
Scenario

promise(4)
Outline

1. Consensus
2. The Part-Time Parliament
3. Single-Decree Paxos
4. Liveness
5. **Multi-Decree Paxos**
6. Paxos Variants
7. Conclusion
Consider Input Ordering in SMR

\[ \text{put}(x,30) \]

\[ X = 3 \]

\[ \text{put}(x,10) \]

\[ X = 3 \]
Paxos Made Moderately Complex

Diagram showing the flow of messages between clients, replicas, and leaders.
Paxos Made Moderately Complex
Paxos Made Moderately Complex
Paxos Made Moderately Complex
Paxos Made Moderately Complex

Diagram showing the interactions between clients, replicas, and leaders in a Paxos system. The diagram highlights the roles of proposers, learners, and acceptors in the consensus process.
Paxos Made Moderately Complex
Paxos Made Moderately Complex
Paxos Made Moderately Complex

replicas | leader | acceptors
---|---|---
\(\rho_1\) | \(\lambda\) | \(\alpha_1\)
\(\rho_2\) | scout | \(\alpha_2\)
p1a | \(\alpha_3\)
p1b | adopted |
| commander | p2a |
p2b | decision |

Prepare
Promise
Paxos Made Moderately Complex

Propose

Promise

Prepare
Paxos Made Moderately Complex

replicas

leader

acceptors

$\rho_1$

$\rho_2$

$\lambda$

scout

p1a

p1b

adopted

commander

p2a

p2b

decision

Prepare

Promise

Propose
Paxos Made Moderately Complex

can both be **preempted** by a higher ballot number being reported
Outline

1. Consensus
2. The Part-Time Parliament
3. Single-Decree Paxos
4. Liveness
5. Multi-Decree Paxos
6. **Paxos Variants**
7. Conclusion
Paxos Variants

- Fast Paxos
- Generalized Paxos
- Disk Paxos
- Cheap Paxos
- Vertical Paxos
- Egalitarian Paxos
- Mencius
- Stoppable Paxos
Paxos in Real Systems

- Chubby
- Google Spanner
- Megastore
- OpenReplica
- Bing
- WANDisco
- XtreemFS
- Doozerd
- Ceph
- Clustrix
- Neo4j
Outline

1. Consensus
2. The Part-Time Parliament
3. Single-Decree Paxos
4. Liveness
5. Multi-Decree Paxos
6. Paxos Variants
7. Conclusion
Conclusion

- **Paxos** is a protocol for solving the **consensus problem** in an **asynchronous** distributed environment with processors that can fail by **crashing**
- A **replicated state machine** can be built by maintaining a **distributed command log** where the command at each position in the log is decided by solving **consensus**
- **Correctly** and **efficiently** implementing a replicated state machine using Paxos is notoriously **difficult**