The ABCDs of Paxos

Consensus: a set of processes *decide* on an input value

Main application: Replicated state machines

Paxos asynchronous consensus algorithm

  **AP** Abstract Paxos: generic, non-local version

  **CP** Classic Paxos: stopping failures, compare-and-swap
  1989: Lamport, Liskov and Oki

  **DP** Disk Paxos: stopping failures, read-write
  1999: Gafni and Lamport

  **BP** Byzantine Paxos: arbitrary failures
  1999: Castro and Liskov

*The paper and slides are at research.microsoft.com/lampson*
Replicated State Machines

Lamport 1978: *Time, clocks and the ordering of events* …

Cast your problem as a deterministic state machine

- Takes client input requests for state transitions, called *steps*
- Performs the steps
- Returns the output to the client.

Make $n$ copies or ‘replicas’ of the state machine.

Use consensus to feed all the replicas the same inputs.

Steps must be deterministic, local to replica, atomic (use transactions)

Recover by replaying the steps (like transactions)

Even a read needs a step, unless the result is “as of step $n$”.
Applications of RSM

Reliable, available data storage system
Airplane flight control

Reflexive applications:
Changing quorums of the consensus algorithm

Issuing a *lease*:
- A lock on part of the state that times out, hence is fault tolerant
- Leaseholder can work on its state without consensus
- Like any lock, a lease can have modes or be hierarchical
The Idea of Paxos

A sequence of views; get a decision quorum in one of them.

Each view $v$ chooses an anchored value $c_v$, equal to any earlier decision.

If a quorum accepts the choice, decision!

Decision is irrevocable, may be invisible, but is any later view’s choice. Choice is changeable, must be visible if there was a decision.

Processes

Actions

Transmit

view change

normal operation
Design Methodology

• Communicate only *stable* predicates: once true always true
• Structure the program as a set of atomic actions
• Make actions as non-deterministic as possible: weakest guards
  Allows more freedom for the implementation
  Makes it clear what is essential
• Separate safety, liveness, and performance
  Safety first, then strengthen guards for liveness and scheduling
• Abstraction functions and simulation proofs
Notation

Subscripts and superscripts for function arguments: \( r_v^a \) for \( r(v, a) \)

State functions used like variables

Actions described like this:

<table>
<thead>
<tr>
<th>Name</th>
<th>Guard</th>
<th>State change</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Close}_v )</td>
<td>( c_v = nil \land x \in anchor_v )</td>
<td>( c_v := x )</td>
</tr>
</tbody>
</table>
Failure Model

A set $M$ of processes (machines)

- A *faulty* process can send arbitrary messages: $F^m$
- A *stopped* process does nothing: $S^m$
- A *failed* process is faulty or stopped. State freezes after failure.

Limits on failure:

- $Z_F = \text{set of sets of processes that can all be faulty}$
- $Z_S = \text{set of sets of processes that can all be stopped}$
- $Z_{FS} = \text{set of sets of processes that can all be failed}$

Examples:

- Fail-stop: $n$ processes, $Z_F = \emptyset$, $Z_S = Z_{FS} = \text{any set of size } < (n+1)/2$
- Byzantine: $n$ processes, $Z_F = Z_S = Z_{FS} = \text{any set of size } < (n+1)/3$
- Intel-Microsoft: $n_I + n_M$ processes, $Z_F = \text{any subset of one side}$
Quorums and Predicates

Quorum set $Q$: set of sets of processes; $q$ in $⇒$ any superset in.

State predicate $g$. Predicate on processes $G$, so $G^m$ is a predicate.
A *stable* predicate once true remains true.

$Q\#G$: A predicate $G$ appears to hold in quorum $Q$, \{ $m \mid G^m \vee F^m$ \} $\in Q$

Shorthand: $Q[r_v^*=x]$ for $Q#(\lambda m \mid r_v^m = x)$.

A *good* quorum is not all faulty: $Q_{\sim F} = \{ q \mid q \notin Z_F \}$

$Q_1$ and $Q_2$ *exclusive*: $Q_1$ quorum for $G \Rightarrow$ no $Q_2$ quorum for its negation.

Means $q_1 \cap q_2 \in Q_{\sim F}$ for any $q_1$ and $q_2$. Example: size $> (n + f)/2$

Lift local $r_v^a=x \Rightarrow \sim(r_v^a=\text{out})$ to global $Q_1[r_v^*=x] \Rightarrow \sim Q_2[r_v^*=\text{out}]$

$Q^+$: ensures $Q$ even after failures: $q^+ - z_{FS} \in Q$ for any $q^+$, $z_{FS}$

A *live* quorum has $Q^+ \neq \{ \}$
Specification for Consensus

type \( X \) = \ldots \quad \text{values to decide on}

var \( d \) : \( (X \cup \{\text{nil}\}) := \text{nil} \) Decision

\( \text{input} \) : \text{set } X := \{\}

Name \quad \text{Guard} \quad \text{State change}

Input(\( x \)) \quad \text{input} := \text{input} \cup \{x\}

\text{Decision: } X \quad d \neq \text{nil} \quad \rightarrow \text{ret } d

\text{Decide} \quad d = \text{nil} \land x \in \text{input} \rightarrow d := x
The Idea of Paxos

A sequence of views; get a decision quorum in one of them.

Each view $v$ chooses an anchored value $c_v$: equals any earlier decision.

If a quorum accepts the choice, decision!

Decision is irrevocable, may be invisible, but is any later view’s choice. Choice is changeable, must be visible to Anchor if there was a decision.

Processes

\[
\begin{array}{cccccccc}
    a & a & a & a & a & a & a & a \\
    a & a & a & a & a & a & a & a \\
    a & a & a & a & a & a & a & a \\
\end{array}
\]

Actions

Start; Input;
Close$^a$ Anchor Choose Accept$^a$ Finish$^a$; STEP$^a$

Transmit

\[
\begin{array}{cccccccc}
    r^a & INPUT & c_v & r_v^a & OUTPUT \\
\end{array}
\]

view change normal operation
Abstract Paxos—AP: State

State

<table>
<thead>
<tr>
<th>Non-local Agents’</th>
<th>State functions</th>
<th>View is</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_v$</td>
<td>$r_v^1 d^1$</td>
<td>$Q_{dec}[r_v^* = x]$</td>
</tr>
<tr>
<td>$input$</td>
<td>$r_v^2 d^2$</td>
<td>$Q_{out}[r_v^* = out]$</td>
</tr>
<tr>
<td>$active_v$</td>
<td>$r_v^3 d^3$</td>
<td>else</td>
</tr>
</tbody>
</table>

$Q_{dec}$ and $Q_{out}$ exclusive

$var = const$ is stable for all these except $input$, and $x \in input$ is stable.
AP: Data Flow

View change to later views

\[ r_u^a = \text{nil} \rightarrow \text{x\in anchor}_v \]

Choose \( v \)

\[ c_v = x \rightarrow r_v^a = c_v \rightarrow d^a = r_v \]

Accept \( v \)

Choose \( v \)

Finish \( v \)

Client \( \text{INPUT}(x) \rightarrow x\in \text{input} \)

Processes

Actions

Transmit

\[ r^a \]

\[ \text{INPUT} \quad c_v \quad r_v^a \quad \text{OUTPUT} \]

View change

Normal operation
Example

<table>
<thead>
<tr>
<th></th>
<th>$c_v$</th>
<th>$r_v^a$</th>
<th>$r_v^b$</th>
<th>$r_v^c$</th>
<th></th>
<th>$c_v$</th>
<th>$r_v^a$</th>
<th>$r_v^b$</th>
<th>$r_v^c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>View 1</td>
<td>7</td>
<td>7</td>
<td>out</td>
<td>out</td>
<td>8</td>
<td>8</td>
<td>out</td>
<td>out</td>
<td></td>
</tr>
<tr>
<td>View 2</td>
<td>8</td>
<td>out</td>
<td>8</td>
<td>out</td>
<td>9</td>
<td>9</td>
<td>out</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>View 3</td>
<td>9</td>
<td>out</td>
<td>out</td>
<td>9</td>
<td>9</td>
<td>out</td>
<td>out</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

(input $\cap$ anchor$_4$) = \{7, 8, 9\} seeing $a$, $b$, $c$

$\supseteq$\{8\} seeing $a$, $b$

$\supseteq$\{9\} seeing $a$, $c$ or $b$, $c$

{9} no matter what quorum we see

Two runs of AP with
agents $a$, $b$, $c$,
two agents in a quorum,
$input$ = \{7, 8, 9\}
Anchoring

\textbf{Invariant} \quad r_v = x \land r_u = x' \Rightarrow x = x'

\begin{align*}
= \quad & \forall x', \ u \mid r_v = x \land r_u = x' \Rightarrow x = x' \\
= \quad & r_v = x \Rightarrow (\forall u < v, x' \neq x \mid \sim Q_{\text{dec}}[r_u^* = x']) \\
\Leftarrow r_v = x \Rightarrow (\forall u < v \mid Q_{\text{out}}[r_u^* \in \{x, \text{out}\}])
\end{align*}

\textbf{sfunc anchor}_v

\begin{align*}
= \quad & \{x \mid (\forall u < v \mid Q_{\text{out}}[r_u^* \in \{x, \text{out}\}])\} \\
= \quad & \{x \mid (\forall w \mid v_0 = w < u \Rightarrow Q_{\text{out}}[r_w^* \in \{x, \text{out}\}])\} \\
& \cap \{x \mid Q_{\text{out}}[r_u^* \in \{x, \text{out}\}]\} \\
& \cap \{x \mid (\forall w \mid u < w < v \Rightarrow Q_{\text{out}}[r_w^* \in \{x, \text{out}\}])\}
\end{align*}

\begin{align*}
= \quad & \text{anchor}_u \cap \{x \mid Q_{\text{out}}[r_u^* \in \{x, \text{out}\}]\}) \text{ if } \text{out}_{u,v} \\
\supseteq \quad & \text{if } \text{out}_{v_0,v} \text{ then } X \text{ elseif } \text{out}_{u,v} \land r_u^a = x \text{ then } \{x\} \text{ else } \{\}
\end{align*}

where \text{out}_{u,v} = (\forall w \mid u < w < v \Rightarrow r_w = \text{out})
**AP: Algorithm**

- **Start**
  - $u < v$ too slow
  - $\rightarrow \text{active}_v := \text{true}$
  - $\rightarrow \text{for all } u < v \text{ do}$
  - $\text{if } r_u^a = \text{nil}$
  - $\text{then } r_u^a := \text{out}$
  - $\text{post } u < v \Rightarrow r_u^a \neq \text{nil}$

- **Close**
  - $\text{active}_v$

- **Anchor**
  - $\text{anchor}_v = \text{anchor}_u \cap \{ x \mid Q_{out}[r_u^* \in \{ x, \text{out} \}] \}$

- **Choose**
  - $c_v^a = \text{nil}$
  - $\land x \in \text{input} \cap \text{anchor}_v$
  - $\rightarrow c_v := x$

- **Accept**
  - $r_v^a = \text{nil}$
  - $\land c_v \neq \text{nil}$
  - $\rightarrow r_v^a := c_v$; **Close**

- **Finish**
  - $r_v \in X$
  - $\rightarrow d_v^a := r_v$

**to later views**

$r_u^a = \text{nil}$
- $\rightarrow \text{Close}_v$
- $x \in \text{anchor}_v$
- $\rightarrow \text{Choose}_v$
- $c_v := x$
- $\rightarrow \text{Accept}_v$
- $r_v^a := c_v$
- $\rightarrow \text{Finish}_v$
- $d_v^a := r_v$
**AP: Liveness**

Choose $v$ must see an element of $\text{input} \cap \text{anchor}_v$.

Recall $\text{anchor}_v$

$$\text{anchor}_v = \text{anchor}_u \cap \{x \mid Q_{out}[r_u \in \{x, \text{out}\}]]\} \quad \text{if } \text{out}_{u,v}$$

$$\supseteq \begin{cases} \text{if } \text{out}_{v0,v} \text{ then } X \quad \text{elseif } \text{out}_{u,v} \land r_u^a = x \text{ then } \{x\} \quad \text{else } \{\} \end{cases}$$

After $\text{Close}^a_v$, an OK agent $a$ has $r_u^a \neq \text{nil}$ for all $u < v$.

So if $Q_{out}$ is live, we see either $u < v$ is out, or $r_u^a = x$ for some OK $a$. But $r_u^a = c_u \in \text{input} \cap \text{anchor}_u$

If we know $a$ is OK, then $r_u^a$ is what we want

With faults (in BP), we might not know.

But if $\text{anchor}_u$ is visible, that is enough.

Still not live if new views start too fast.
Optimizations

Fixed-size agent state:

\[ r_w^a = \quad \text{don’t know} \quad x_{last}^a \quad \text{out} \quad \text{nil} \]

view \( v_0 \)

Successive steps:

Because \( anchor_v \) doesn’t depend on \( input \), can compute it for lots of steps at once.

This is called a view change

One view change is enough for any number of steps

Can batch steps, with one Paxos/batch.

Can run steps in parallel, subject to external consistency.
Disk Paxos—DP

The goal—Replace the conditional writes in Close and Accept with simple writes.

\[ \text{Accept}_v^a \quad r_v^a = \text{nil} \land c_v \neq \text{nil} \quad \rightarrow r_v^a := c_v; \ Close_v^a \]

The idea—Replace \( r_v^a \) with \( rx_v^a \) and \( ro_v^a \).

\[ \text{Accept}_v^a \quad c_v \neq \text{nil} \quad \rightarrow rx_v^a := c_v; \ Close_v^a \]
\[ \text{Close}_v^a \quad \text{active}_v \quad \rightarrow \text{for all } u < v \text{ do } ro_u^a := \text{out} \]

Proof: Keep \( r_v^a \) as a history variable. Abstract it to AP’s \( r_v^a \).
This invariant makes it work (sometimes with an extra view).

\[
\begin{align*}
rx_v^a &= \land \quad ro_v^a &= \Rightarrow \quad r_v^a \\
nil &= \text{nil} \\
nil &= \text{out} \\
x &= \text{nil} \\
x &= \text{out} \\
x &= \neq \text{nil}
\end{align*}
\]
Communication

A process has knowledge $T$ of stable non-local facts

$$g@m = (T^m \Rightarrow g)$$

We transmit these facts (note that transmitter $k$ may be failed):

$$\text{Transmit}^{k,m}(g) \quad g@k \lor F^k \rightarrow T^m := T^m \land (g@k \lor F^k) \quad \text{post } (g@k \lor F^k)@m$$

A faulty $k$ can transmit anything:

A fact known to a $Q_{-F}^+$ quorum is henceforth known to a $Q_{-F}$ quorum of OK agents, and therefore eventually known to everyone.

$$\text{Broadcast}^m(g) \quad Q_{-F}^+[g@*] \land OK^m \rightarrow T^m := T^m \land g \quad \text{post } g@m$$

Implement $\text{Transmit}^{k,m}$ by sending messages. It’s fair if $k$ is OK. This works because the facts are stable.
Classic Paxos—CP

The goal—Tolerate stopped processes

The idea—Agents are the same as in AP. Use a primary process to:
- Implement Choose
- Compute an estimate $re_v$ of $r_v$
- Relay facts among the agents
- Do all the scheduling.

So the primary sends $active_v$ to agents to enable $Close_v$, collects $r^a$, computes anchor, gets inputs, does Choose, sends $c^p$ to agents, collects $r^a$ again to compute $re_v$, and sends $d$.

$$Choose^p \quad \quad active^p \land c^p = nil \quad \rightarrow c^p := x \quad \land x \in input^p \cap anchor^p$$

Must have only one $c^p$ per view. Get this with
- At most one primary per view, and
- Primary chooses at most once per view
Byzantine Paxos—BP

The goal—Tolerate faulty processes

The idea—To ensure one $c_v$, a self-exclusive quorum $Q_{ch}$ chooses it

Still have a primary to propose $c_v$; an OK agent only chooses this

Primary’s proposal should be anchored and input at agents
A faulty primary can stop its view from deciding

Every agent needs an estimate $ce_v^a$ of $c_v$ and an estimate $re_v^a$ of $r_v$

Invariant: The estimates either are $nil$ or equal the true values.

Every agent also needs its own input$^a$

$$\text{abstract } c_v = \begin{cases} x & \text{if } Q_{ch}[c_v^* = x] \\ \text{nil} & \text{else} \end{cases}$$

$$\text{sfunc } ce_v^a = \begin{cases} x & \text{if } (Q_{ch}[c_v^* = x])@a \\ \text{nil} & \text{else} \end{cases}$$

$$anchor_v^a = anchor_u \cap \{x \mid Q_{out}[r_u^* \in \{x, out\}]@a\} \quad \text{if } out_{u,v}^a$$

$$anchor_v^p = \{x \mid Q_{~F}^+[x \in anchor_v^*]@p\}$$
CP and BP

**CP**

Processes

![Diagram](cpProcesses.png)

Actions

- Start$^p$
- Input$^p$
- Close$^p$
- Close$^a$
- Anchor$^p$
- Anchor$^a$
- Choose$^p$
- Choose$^a$
- Accept$^p$
- Accept$^a$
- Finish$^p$
- Finish$^a$

Transmit

- active$_v$
- $r^a$
- INPUT
- $c^p$
- $r_v^a$
- OUTPUT
- $r_v$,$_v$

Messages

- $1\rightarrow n^*$
- $n^*\rightarrow 1$
- $1\rightarrow 1$
- $1\rightarrow n^*$
- $1\rightarrow n^*$

**BP**

Processes

![Diagram](bpProcesses.png)

Actions

- Start$^a$
- Input$^a$
- Close$^a$
- Anchor$^a$
- Anchor$^p$
- Anchor$^a$
- Choose$^p$
- Choose$^a$
- Accept$^a$
- Finish$^a$

Transmit

- $r^a$
- $c^a$
- anchor$_v$
- INPUT
- $c_v^p$
- $c_v^a$
- $r_v^a$
- OUTPUT
- $n\rightarrow n$
- $n^*\rightarrow 1$
- $1\rightarrow n$
- $1\rightarrow n^*$
- $n\rightarrow n$
- $n\rightarrow 1$
Liveness of BP

Choose must see an element of $\text{input} \cap \text{anchor}_v$.

Recall $\text{anchor}_v \supseteq \text{anchor}_u \cap \{x \mid Q_{out}[r_u^* \in \{x, \text{out}\}]\}$

After $\text{Close}_v^a$, an OK agent $a$ has $r_u^a \neq \text{nil}$ for all $u < v$.

So if $Q_{out}$ is live, we see either $u < v$ is out, or $r_u^a = x$ for some OK $a$.

But $r_u^a = c_u \in \text{input} \cap \text{anchor}_u$

Unfortunately, we don’t know whether $a$ is OK.

But we do have $Q_{ch}[c_u^* = x]$, hence $Q_{ch}[(x \in \text{anchor}_u)@a]$.

So if $Q_{ch}$ is live, $x \in \text{anchor}_u$ is broadcast, which is enough.

So either we eventually see all previous views out, or we see $x \in \text{anchor}_u$ and all views between $u$ and $v$ out.

A faulty client can wreck a view by not sending input to all agents.
Conclusion

Paxos is a practical protocol for fault-tolerant asynchronous consensus.

Paxos is efficient in replicated state machines, which are the best mechanism for most fault-tolerant systems.

Paxos works in a sequence of views,

- Each view chooses a value and then seeks a decision quorum.
- A later view chooses any possible earlier decision

Abstract Paxos chooses a consensus value non-locally, and then decides by local actions of the agents.

- The agents are read-modify-write memories.
- Disk Paxos generalizes this to read-write memories.

Classic Paxos uses a primary process to choose.

Byzantine Paxos uses a primary to propose, a quorum to choose.