The Part-Time Parliament

Leslie Lamport, 1998

Carlos Eduardo B. Bezerra

Faculty of Informatics
Università della Svizzera italiana

May 25th, 2011

Outline

1. Introduction
   - A long way to Paxos
   - The consensus problem

2. The Paxos Algorithm
   - Notations used
   - Consistency conditions
   - Proof sketch
   - Preliminary protocol

Introduction

Interesting information about 'The Part-Time Parliament':

- the algorithm was first described in 1989 as a technical report [1]
- the paper was submitted in 1990. . . and accepted in 1998
- several other publications, such as [2, 3, 4], had explaining it as their main purpose. . .

The paper mostly uses an analogy with the fictional ancient island of Paxos, where:

- The parliament's (group's) legislators (processes) and their messengers (communication channels) may be unpredictably absent;
- There may be several ballots for each decree to be passed (rounds in each consensus instance);
- Legislators – and messengers – may be lazy or absent, but not dishonest (non-Byzantine failure model*)

* The algorithm was extended later, however, to tolerate Byzantine failures [5]

Definition

Consensus is the process of agreeing on one single result among a group of participants. Assuming a collection of processes that can propose values, a consensus algorithm ensures that:

- only a single value among the proposed ones may be chosen;
- if no value has been proposed, then no value is chosen;
- a process never learns that a value has been chosen unless it really has been.

The Paxos algorithm
Some notations used

A value is chosen after a series of numbered **ballots**. For each ballot B, we have that:

- B_{bal} is its ballot number;
- B_{qrm} is a nonempty finite set of processes (quorum);
- B_{vot} is the set of processes who voted in B – each process may decide to participate (vote) or not in B;
- B_{val} is the value being voted on in that ballot.

A ballot is said to be *successful* when $B_{qrm} \subseteq B_{vot}$ – which means that every quorum member voted.

The Paxos algorithm

Some more notations

In each ballot, each process can cast a vote (accept or not a given proposed value). A vote v has the components:

- v_p, which is the process p who cast it;
- v_{bal}, the number of the ballot for which it was cast;
- v_{val}, a value voted on.

For the set β of ballots, the set $Votes(\beta)$ is defined as
$\{v : \exists B \in \beta \mid v_p \in B_{vot} \wedge v_{bal} = B_{bal} \wedge v_{val} = B_{val} \}$

The Paxos algorithm

Still more notations...

We then define *MaxVote(b, p, β)* as the vote v with highest v_{bal} in $\{v \in Votes(\beta) : (v_p = p) \wedge (v_{bal} < b)\} \cup \{null\}$,
where *null* means that p didn't vote in any $B \in \beta$ ($null_{bal} = -\infty$)

We also define *MaxVote(b, Q, β)* as the vote v, such that v_{bal} is the highest among all *MaxVote(b, p, β)* with $p \in Q$

With such notations, we can specify the conditions which ensure consistency and allow for progress.

Consistency conditions

Consistency is guaranteed if the following conditions are ensured:

- $B1(\beta)$
  $\forall B, B' \in \beta : (B \neq B') \Rightarrow (B_{bal} \neq B'_{bal})$
- $B2(\beta)$
  $\forall B, B' \in \beta : B_{qrm} \cap B'_{qrm} \neq \emptyset)$
- $B3(\beta)$
  $\forall B, B' \in \beta :$
  $(MaxVote(B_{bal}, B_{qrm}, \beta)_{bal} \neq -\infty) \Rightarrow$
  $\Rightarrow (B_{val} = MaxVote(B_{bal}, B_{qrm}, \beta)_{val})$

Proof sketch

Lemma. If $B1(\beta)$, $B2(\beta)$ and $B3(\beta)$ hold, then

$$((B_{qrm} \subseteq B_{vot}) \wedge (B'_{bal} > B_{bal})) \Rightarrow (B'_{val} = B_{val})$$

for any B, B' in β.

Proof of Lemma. Let $\Psi(B,\beta)$ be defined as:

$$\Psi(B,\beta) = \{B' \in \beta : (B'_{bal} > B_{bal}) \wedge (B'_{val} \neq B_{val})\}$$

Assume that $B_{qrm} \subseteq B_{vot}$. By contradiction, assume that $\Psi(B,\beta) \neq \emptyset$.

(1) As $\Psi(B,\beta) \neq \emptyset$, we pick $C \in \Psi(B,\beta)$, such that
$C_{bal} = min\{B'_{bal} : B' \in \Psi(B,\beta)\}$

(2) From (1) and from the definition of $\Psi(B,\beta)$, we have that $C_{bal} > B_{bal}$

Proof sketch

continuing. . .

(3) As $B_{qrm} \subseteq B_{vot}$ and $\forall B, B' \in \beta : B_{qrm} \cap B'_{qrm} \neq \emptyset$, we have that $B_{vot} \cap C_{qrm} \neq \emptyset$

(4) As $C_{bal} > B_{bal}$ and $B_{vot} \cap C_{qrm} \neq \emptyset$, we have that $MaxVote(C_{bal}, C_{qrm}, \beta)_{bal} \geq B_{bal}$

(5) From (4) and the definition of $MaxVote(C_{bal}, C_{qrm}, \beta)$, we know that such vote exists in $Votes(\beta)$ (i.e. it is not *null*).

(6) From (5) and $B3(\beta)$, we have that $MaxVote(C_{bal}, C_{qrm}, \beta)_{val} = C_{val}$

(7) From (6), and as $C_{val} \neq B_{val}$ (from $\Psi(B, \beta)$), we have that $MaxVote(C_{bal}, C_{qrm}, \beta)_{val} \neq B_{val}$

Proof sketch

continuing...

(8) As $MaxVote(C_{bal}, C_{qrm}, \beta)_{bal} \geq B_{bal}$ (from (4)) and $MaxVote(C_{bal}, C_{qrm}, \beta)_{val} \neq B_{val}$ (from (7)), we have that $MaxVote(C_{bal}, C_{qrm}, \beta)_{bal} > B_{bal}$

(9) From (8), and since $MaxVote(C_{bal}, C_{qrm}, \beta)_{val} \neq B_{val}$, we have that $MaxVote(C_{bal}, C_{qrm}, \beta) \in Votes(\Psi(B, \beta))$

(10) By the definition of $MaxVote(C_{bal}, C_{qrm}, \beta)$, we have that $MaxVote(C_{bal}, C_{qrm}, \beta)_{bal} < C_{bal}$

(11) From (10), we have that $\exists B' : B' \in \Psi(B, \beta) \wedge B'_{bal} < C_{bal}$, which means that

$$C_{bal} \neq min\{B'_{bal} : B' \in \Psi(B, \beta)\}$$

which is a contradiction with (1). $\square$

Proof sketch

continuing...

With the lemma, we show that, for a given set β of ballots which hold $B1$, $B2$ and $B3$, then any two successful ballots decided the same value.

Theorem 1. If $B1(\beta)$, $B2(\beta)$ and $B3(\beta)$ hold, then

$$((B_{qrm} \subseteq B_{vot}) \wedge (B'_{qrm} \subseteq B'_{vot})) \Rightarrow (B'_{val} = B_{val})$$

for any B, B' in β.

Proof of Theorem. If $B'_{bal} = B_{bal}$ then $B1(\beta)$ implies $B' = B$. If $B'_{bal} \neq B_{bal}$, the theorem follows immediately from the lemma. $\square$

Proof sketch

continuing. . .

If there are enough correct (non-faulty) processes, then a new ballot may be conducted while $B1$, $B2$ and $B3$ are preserved.

Theorem 2. Let b be a ballot number, and let Q be a set of processes such that $b > B_{bal}$ and $Q \cap B_{qrm} \neq \emptyset$, for all $B \in \beta$. If $B1(\beta)$, $B2(\beta)$ and $B3(\beta)$ hold, then there is a ballot B' with $B'_{bal} = b$ and $B'_{qrm} = B'_{vot} = Q$ such that $B1(\beta \cup \{B'\})$, $B2(\beta \cup \{B'\})$ and $B3(\beta \cup \{B'\})$ hold.

Proof of Theorem.

- $B1(\beta \cup \{B'\})$ follows from $B1(\beta)$ and the fact that $B'_{bal} = b$, and the assumption about b.
- $B2(\beta \cup \{B'\})$ follows from $B2(\beta)$ and the fact that $B'_{qrm} = Q$, and the assumption about Q.
- For $B3(\beta \cup \{B'\})$, if $MaxVote(b, Q, B) = -\infty$ then let B'_{val} be any value; otherwise, let $B'_{val} = MaxVote(b, Q, B)_{val}$. $B3(\beta \cup \{B'\})$ then follows from $B3(\beta)$. $\square$

Preliminary protocol

A protocol can be derived directly from $B1(\beta)$, $B2(\beta)$ and $B3(\beta)$, so that all these properties are maintained throughout its execution and a value may* be agreed upon.

- To maintain $B1(\beta)$, whenever p starts a new ballot B, it assigns a sequence number *seq*, which was never used by p. To avoid collisions, p's unique id is appended. Then $B_{bal} = (seq, p_{id})$.
- To maintain $B2(\beta)$, each ballot B has B_{qrm} as the majority set of the processes. As it is impossible to choose two disjoint majority sets from the same group*, $B2(\beta)$ is satisfied.
- To maintain $B3(\beta)$, before initiating a ballot B, p has to find out what is $MaxVote(B, B_{qrm}, \beta)$, which is done by finding the $MaxVote(B, q, \beta)$ for each q in B_{qrm}.
  - If $MaxVote(B, B_{qrm}, \beta) = null$, then B_{val} can be anything; otherwise $B_{val} = MaxVote(B, B_{qrm}, \beta)_{val}$.

* If half of the processes is faulty, consistency is still respected and there is no deadlock, but the ballot B does not succeed

Preliminary protocol

Phase 1) Find $MaxVote(B, B_{qrm}, \beta)_{val}$

Phase 1a) p picks a ballot number b and sends $NextBallot(b)$ to other processes

Phase 1b) each process q responds with $LastVote(b, vote)$ to p, where *vote* is the vote cast by q with the highest ballot number less than b (or *null*)

Phase 2) Execute the ballot B

Phase 2a) after receiving $LastVote(b, vote)$ from a majority Q of the processes, p initiates the ballot with number b and value *val* such that $B3(\beta)$ is maintained, by sending $BeginBallot(b, val)$ to every process in Q

Phase 2b) upon receiving $BeginBallot(b, val)$, a process will not vote on it if it has already sent $LastVote(b', val')$, such that $b' > b$; otherwise, it sends $Voted(b, q)$ to p and records this vote in its memory

Preliminary protocol

Phase 3) Announce the result of consensus

Phase 3a) if p receives $Voted(b, q)$ from a majority Q of processes, then it sends $Success(val)$ to all processes

Phase 2b) upon receiving $Success(val)$, the process learns that the group achieved consensus on val.

Summary

- The **first main message** of your talk in one or two lines.

- Outlook
  - Something you haven't solved.
  - Something else you haven't solved.

References

[1] Lamport, L. **The part-time parliament**, Technical Report 49, Systems Research Center, Digital Equipment Corp., 1989

[2] Lamport, L. **Paxos Made Simple**, ACM SIGACT News, 2001

[3] De Prisco, R., Lampson, B., Lynch, N. **Revisiting the Paxos Algorithm**, Distributed Algorithms, 1997

[4] Lampson, B. **How to build a highly available system using consensus**, Distributed Algorithms, 1996

[5] Castro, M., Liskov, B., **Practical Byzantine Fault Tolerance**, Operating Systems Design and Implementation, 1999