Emmanuelle Anceaume

The consensus problem in synchronous environments

http://people.irisa.fr/Emmanuelle.Anceaume/
Content of this lesson

- Consensus problem in synchronous environments
  - algorithm that tolerates crash failures
  - lower bound on the number of rounds
  - algorithm that tolerates Byzantine failures

- Consensus problem in asynchronous environments
  - Impossibility result even in a presence of at most one single crash failure
Consensus
Informal specification

- In this problem processors are trying to reach a consensus on a value.
- Each processor initially proposes a value \( v \) taken from a given set of value \( V \).
- At the end of the protocol, all processors agree on a single value, called the decided value, or decision.
- Distributed systems: a set of $n$ processors that aim at solving a given task
  - $\Pi = \{p_1, \ldots, p_n\}$.
- Message-passing model: processors communicate and synchronize by exchanging messages
- Communication graph is complete
Among the $n$ processes, up to $f$ processors are faulty
- a faulty processor is a processor whose functionality is incorrect
- there are different ways for a processor to be incorrect: from crash failures to Byzantine failures
  - a crash processor stops operating, but does not perform wrong operations (e.g., it cannot deliver messages which were not sent or write an erroneous value in a register)
  - a Byzantine processor may behave arbitrarily. We cannot make any assumptions on the way it behaves

It is not known in advance which processors are faulty
The system is synchronous: an execution is partitioned into rounds.

In each round:

- Each non-faulty processor executes one computational step, sends messages to its neighbors, and delivers all the messages sent to it during the round.
- Each faulty processor may not complete its round in its entirety (e.g., its messages may not have been received by all its neighbors). Once a processor has crashed it stops forever its execution.
Message-passing synchronous systems

Round $i$  Round $i+1$
$p_1$
$p_2$
$\ldots$
$p_n$
time

crashed processors do not execute anymore
Message-passing synchronous systems

A Byzantine processor behaves arbitrarily
Each processor $p_i$ has special state components

- $x_i$: the input
- $y_i$: the output (also called decision)

Initially, $x_i$ holds a value, and $y_i$ is undefined. Any assignment to $y_i$ is irreversible.
A solution to the Consensus problem must guarantee the following:

- **Termination**: \( y_i \) is eventually assigned a value for every non-faulty processor \( p_i \).
- **Agreement**: If \( y_i \) and \( y_j \) are assigned, then \( y_i = y_j \) for all non-faulty processors \( p_i \) and \( p_j \).
- **Validity**: If for some value \( v \), \( x_i = v \) for all processors \( p_i \), and \( y_i \) is assigned for some non-faulty processor \( p_i \) then \( y_i = v \).

Validity property: makes a solution to the consensus problem not trivial.
A simple consensus algorithm in the crash model

```
propose(v_i) { // algorithm run by process p_i
Initially V = v_i

1: round k, 1 ≤ k ≤ f + 1
2: send {v ∈ V: p_i has not already sent v} to all processors
3: receive S_j from p_j, 0 ≤ j ≤ n − 1, j ≠ i
4: V := V ∪ \bigcup_{j=0}^{n−1} S_j
5: y = \min(V)
6: return decide(y)
}
```
A simple consensus algorithm: correctness

propose($v_i$) { // algorithm run by process $p_i$

Initially $V = v_i$

1: round $k$, $1 \leq k \leq f + 1$
2: send $\{v \in V: p_i \text{ has not already sent } v\}$ to all processors
3: receive $S_j$ from $p_j$, $0 \leq j \leq n - 1$, $j \neq i$
4: $V := V \cup \bigcup_{j=0}^{n-1} S_j$
5: $y = \min(V)$
6: return decide($y$)
}

- Termination: Each non-faulty process must eventually decide on a value
  - from the code, the algorithm requires $f + 1$ rounds
A simple consensus algorithm: correctness

```plaintext
propose(v_i) { // algorithm run by process p_i
Initially V = v_i

1: round k, 1 ≤ k ≤ f + 1
2: send \{v ∈ V: p_i has not already sent v\} to all processors
3: receive S_j from p_j, 0 ≤ j ≤ n − 1, j ≠ i
4: V := V ∪ \bigcup_{j=0}^{n-1} S_j
5: y = \min(V)
6: return decide(y)
}
```

- **Validity:** If all the non-faulty processes have the same initial value, then the agreed upon value by all the non-faulty processes must be that same value.
  - Processors do not send fictitious values in this failure model. For all i, j, if the initial value v_i is identical to v_j then the only value that can be decided is v_i = v_j.
A simple consensus algorithm: correctness

- Agreement: If $y_i$ and $y_j$ are assigned, then $y_i = y_j$ for all non-faulty processors $p_i$ and $p_j$.

Lemma

In every execution, at the end of round $f + 1$, $V_i = V_j$ for every nonfaulty $p_i$ and $p_j$.

Proof: It suffices to show that if $x \in V_i$ at the end of round $f + 1$ then $x \in V_j$ at the end of round $f + 1$, for all nonfaulty $p_i$ and $p_j$.

Let $r = \text{1rst round at which } x \in V_i$ for any nonfaulty $p_i$ (L4) (if $x$ is already in $V_i$, $r = 0$)

- if $r \leq f$ then at round $r + 1$ $p_i$ broadcasts $x$ to each $p_j$ (L2), and $p_j$ adds $x$ to $V_j$ if not already done (L4)
A simple consensus algorithm: correctness

Proof (cont’d)

- Otherwise, suppose that $r = f + 1$.
  - let $p_j$ be a nonfaulty such that $p_j$ receives $x$ for the first time in round $f + 1$. There must have a chain of $f + 1$ processors such that $x$ is transferred to $p_j$
    - $p_{i_1}$ sends $x$ to $p_{i_2}$ in round 1
    - $\ldots$
    - $p_{i_f}$ sends $x$ to $p_{i_{f+1}}$ in round $f$, and
    - $p_{i_{f+1}}$ sends $x$ to $p_j$ in round $f + 1$
  - by construction each processor sends $x$ only once
  - thus $p_{i_1}, p_{i_2}, \ldots, p_{i_{f+1}}$ are $f + 1$ distinct processors
  - thus there must be at least one correct processor among them
  - thus this processor must have stored $x$ in round $\leq f < r$
  - which contradicts the assumption that $r$ is minimal

- As a consequence, nonfaulty processors have the same set $V$ in L5
- And thus decide on the same value
A simple consensus algorithm: correctness

Illustration of case 2 of Lemma 1. f=3
A simple consensus algorithm: complexity

- There are $f + 1$ rounds, where $f < n$
- The number of messages sent in each round is at most $n^2$
- Hence the total number of messages is $O((f + 1)n^2)$. 
We present a lower bound of $f + 1$ on the number of rounds required for reaching consensus in the presence of $f$ crash failures.

We focus on binary consensus - fine for a lower bound.

The intuition behind the lower bound is that if processors decide too early, they cannot distinguish between executions in which they should make different decisions.

We will consider failure-sparse executions, i.e., executions in which there is at most one crash per round.

Stretching out the failures over more rounds increases the amount of time in which there is uncertainty about failures.
Synchronous system

We consider a synchronous system consisting of a set of processors $p_1, \ldots, p_n$ and communication channels

- Each $p_i$ has a one-bit input register $x_{p_i}$, and output register $y_{p_i}$ with values in $\{0, 1, b\}$
- The state of $p_i$ comprises the value of $x_{p_i}$, the value of $y_{p_i}$ (and its program counter, and its internal storage...)
- Initial state of $p_i : x_{p_i} = 0$ or $x_{p_i} = 1$ and $y_{p_i} = b$
- Decision states : $y_{p_i} = 0$ or $y_{p_i} = 1$
- Transition function
  - deterministic
  - cannot change the decision value ($y_{p_i}$ is writable only once)
Definition (View of a processor)

Let $\alpha$ be an execution and let $p_i$ be a processor. The view of $p_i$, denoted by $\alpha|_{p_i}$, is the subsequence of computation and message delivery events that occur in $\alpha$ at $p_i$ together with $p_i$’s initial configuration.

Definition (Similar executions)

Let $\alpha_1$ and $\alpha_2$ be two executions and let $p_i$ be a processor that is nonfaulty in both $\alpha_1$ and $\alpha_2$. $\alpha_1$ is similar to $\alpha_2$ with respect to $p_i$, denoted by $\alpha_1 \sim_{p_i} \alpha_2$, if $\alpha_1|_{p_i} = \alpha_2|_{p_i}$.
Some preliminary definitions

A key notion is the set of decisions that can reached from a particular configuration

**Definition (Valence of a configuration)**

The valence of a configuration $C$ is the set of all values that are decided by nonfaulty processors in a configuration reachable from $C$

- $C$ is univalent if this set contains a single value
  - 0-valent if the value is 0
  - 1-valent if the value is 1
- bivalent if the set contains both 0 and 1
Proof of the lower bound

**Theorem**

*Any consensus algorithm $A$ for $n$ processors, resilient to $f$ crash failures, requires at least $f + 1$ rounds in some execution for all $n \geq f + 2$*

Proof. Let $A$ be any consensus algorithm for $n$ processors and $f$ crash failures

1. we first show that there exists an $(f - 1)$-round execution of $A$ in which the configuration at the end is undecided
   - by induction on the round number
2. we then show that with just one more round it is not possible for the processors to decide explicitly
3. thus at least $f + 1$ rounds are required for deciding
Lemma (1) : Bivalent initial configuration(s)

Algorithm $A$ has a bivalent initial configuration.
Proof: By contradiction. Suppose that all the initial configurations are univalent. By the validity property,
- initial configurations such that all inputs are 0 is 0-valent
- initial configurations such that all inputs are 1 is 1-valent

We can order initial configurations in a chain of configurations, where two configurations are next to each other if they differ by only one value
- the difference between two adjacent configurations is the starting value of a one processor

```
[0,0,...,0]  [1,0,...,0]  [1,1,...,0]  [1,...,1,0,0,...,0]  [1,...,1,1,0,...,0]  [1,1,...,1]
  0           1           2          ●●●●●●  i           i+1          ●●●●●●  n
```
Proof of Lemma (1)
Proof of Lemma (1)
Proof of Lemma (1)
Proof of Lemma (1)
Proof of Lemma (1)

In $\sigma$, processor $p_i$ takes no step
So its initial value cannot be observed by someone else

All processors must eventually decide
(failure tolerant protocol)

Since $C_0$ is 0-valent the decision state is 0
Proof of Lemma (1)

In \( \sigma \) processor \( p_i \) fails initially (i.e., does not receive nor send messages).

So its initial value cannot be observed.

All processors must eventually decide (1-failure tolerant protocol)

Since \( C_0 \) is 0-valent the decision state is 0.

Run \( \sigma \) can be made from \( C_1 \) too since no processor has ever heard about \( p_i \).

Thus all the processors (except \( p_i \)) should reach the "0" deciding state.

This is a contradiction since by assumption \( C_1 \) is a "1"-valent configuration.
Proof of Lemma (1)

- So this result contradicts the fact that the outcome of the consensus algorithm is uniquely predetermined by the initial configurations.
- $C_0$ can lead to a "0" decision state or to a "1"-decision state, depending on the pattern of failures.

Initial bivalent configuration

Any synchronous consensus algorithm that is resilient to $f$ crashes has at least one bivalent initial configuration.
We have just shown that there exists an initial bivalent configuration (Lemma 1).

We are now going to show that there exists a \((f - 1)\) round execution of \(\mathcal{A}\) in which the configuration at the end is undecided (Lemma 2).

Then we will show that with just one more round it is not possible for the processors to decide explicitly, and thus \(f\) rounds are not sufficient.
Lemma (2)

For each \( k, 0 \leq k \leq f - 1 \), there is a \( k \)-round execution of \( A \) that ends in a bivalent configuration.

Proof: by induction on the round number

- The base case, \( k = 0 \), follows from Lemma (1).

- Assume that the lemma is true for rounds \( 0 \leq k' \leq f - 2 \), and show that it is true for \( 0 \leq k \leq f - 1 \).

- Let \( \alpha_{k-1} \) be the \((k - 1)\)-round execution ending in a bivalent configuration (exists by the inductive assumption).

- Assume by contradiction that all one-round extensions of \( \alpha_{k-1} \) with at most one crash failure end in 0-valent and 1-valent configurations.
Bivalent configuration

α_{k-1} < f - 1\crash failures
at most 1\crash failure
rounds 1 to k-1

Suppose that \( p_i \) fails to send to the subset of processors \( p_1, \ldots, p_m \) with \( 1 \leq m \leq n \).
Switch from a 1-valent configuration to a 0-valent configuration

The only difference between both $\alpha_j^k$ and $\alpha_{j+1}^k$ is that $p_i$ sends a value to $p_{j+1}$ in $\alpha_j^k$ but not in $\alpha_{j+1}^k$
the number of crashes in $\alpha_k^j$ and in $\alpha_k^{j+1}$ is less than $f$ (since at most $k - 1 \leq f - 2$ processors crash in $\alpha_{k-1}$ and $p_i$ crashes in round $k$)

thus there is still one more processor that can crash without violating the bound $f$

consider both extensions

- $\delta_{k+1}^j$ of $\alpha_k^j$
- $\delta_{k+1}^{j+1}$ of $\alpha_k^{j+1}$

in both extensions, $p_{j+1}$ fails at the beginning of round $k + 1$

$p_{j+1}$ did not get the opportunity of revealing whether or not it received a message from $p_i$ at the beginning of round $k + 1$

both extensions are similar with respect to every nonfaulty processor (since the only difference between them is that $p_i$ sends its information to $p_{j+1}$ in $\delta_k^j$ but not in $\delta_k^{j+1}$, but $p_{j+1}$ failed)
Bivalent configuration

$\alpha_{k-1} < f - 1$ crash failures

1 crash failure

round $k$

$\beta_k = \alpha^0_k$

1-valent

$\alpha^j_k$ does not fail to send to anyone

$p_i$ fails to send to $p_j \ldots p_m$

Contradiction!
Thus there must exist a one-round extension of $\alpha_{k-1}$ ending in a bivalent configuration
we have shown that we necessarily have a \((f - 1)\) round execution ending in a bivalent configuration.

we now show that the \(f\)-th round execution does not necessarily preserve bivalence, but nonfaulty processors cannot determine yet what decision to make, and thus an additional round is necessary.
Lemma (3)

**Lemma**

Let $\alpha_{f-1}$ be an $(f - 1)$-round execution of $A$ that ends in a bivalent configuration, then there exists a one round extension of $\alpha_{f-1}$ in which some non-faulty processor has not decided.

**Proof**
Lemma (3)

Proof

- Let $\beta_f$ be the one-round extension of $\alpha_{f-1}$ in which no failure occurs in round $f$
- If $\beta_f$ ends in a bivalent configuration we are done
- Suppose that $\beta_f$ ends in a univalent configuration (say 1-valent)
- Since the configuration at the end of $\alpha_{f-1}$ is bivalent, some other one-round extension of $\alpha_{f-1}$ must be 0-valent. Let $\gamma_f$ be this extension
- Let $p_i$ be some faulty processor
Lemma (3)

\( \beta_f \) and \( \delta_f \) are similar with respect to \( p_k \). Thus \( p_k \) is either undecided or decide 1 at the end of round \( f \).

\( \gamma_f \) and \( \delta_f \) are similar with respect to \( p_j \). Thus \( p_j \) is either undecided or decide 0 at the end of round \( f \).

Since algorithm A satisfies the agreement Property, it cannot be the case that in \( \delta_f \) both \( p_k \) and \( p_j \) have decided.
Thus Lemmas 2 and 3 together imply the existence of an $f$-round execution in which some nonfaulty processor has not decided. Thus $f + 1$ rounds are required for termination.
Byzantine generals

Several divisions of the Byzantine army are waiting outside an enemy city. Each division is commanded by a general. Generals can communicate with reliable messengers.

Each general should eventually decide (Termination) on a plan, and this plan should be common: attack the city or not (Agreement), and if the generals are unanimous in their initial opinion, then that opinion should be the decision (Validity). Some of the generals may be traitors, and may try to prevent the loyal generals from agreeing.

To do so, traitors send conflicting messages to different generals, falsely report on what they have heard from other generals, and even conspire and form a coalition.
Consensus problem with Byzantine failures

- **Agreement**: All non-faulty processors must agree on the same single value.
- **Validity**: If all the non-faulty processors have the same initial value, then the agreed upon value by all the non-faulty processors must be that same value.
- **Termination**: Each non-faulty processor must eventually decide on a value.
The phase-king algorithm (proposed by Berman and Garay) solves the consensus problem in a synchronous model

- $n > 4f$ processors, $f$ Byzantine processors
- the algorithm requires $f + 1$ phases, each phase made of two rounds
- at each phase, a unique processor plays an asymmetrical role (leader)
Rotating coordinator: allows us to break the symmetry among processors

Each phase is partially under the control of a coordinating processor

The identity of the coordinator is given by the round number (thus each processor knows who is the current coordinator)

Each processor $p_i$ maintains an estimate $est_i$ of the decision value
Two principles:

1. If the occurrence number of the most frequent estimate value passes some threshold, this value will be the decided value.

2. Otherwise, the current coordinator will force an estimate value to be adopted by enough processors, so that case (1) applies.
The phase-king algorithm

The first round of phase $k$, i.e. round $r = 2k - 1$, is an estimate determination:

- Processors exchange their current estimate
- At the end of the round, each processor $p_i$ determines the one it sees the most often, and keeps it in $\text{most}_i\_freq$ (if no majority, a default value $v_\bot$ is used)

The second round of phase $k$, i.e. round $r = 2k$, is an estimate adoption:

- If the estimate of $p_i$ passes the threshold then $p_i$ adopts it as the new estimate
- Otherwise, the coordinator $p_k$ broadcasts its $\text{most}_k\_freq$ to all processors
At most $f$ failures can occur

Thus a sequence of $f + 1$ phases includes necessarily a correct coordinator

So this coordinator will impose its value if up to this phase no value passed the threshold
The phase-king algorithm

propose($v_i$) {// algorithm run by process $p_i$
Initially $est_i \leftarrow v_i$, $\forall j \neq i est_j \leftarrow v_\perp$
$f \leftarrow$ maximum number of malicious processors, $f < \lceil n/4 \rceil$
// Each processor executes the following $f + 1$ phases

1: Round $2k - 1$, $1 \leq k \leq f + 1$
2: send $est_i$ to all
3: receive $v_j$ from $p_j$ and $est_j \leftarrow v_j$ for all responses
4: $maj \leftarrow$ the majority value of $est_1, \ldots, est_n$ ($v_\perp$ if none)
5: $mult \leftarrow$ number of times $maj$ occurs

6: Round $2k$, $1 \leq k \leq f + 1$
7: if $i = k$ then send $maj$ to all
8: receive $king\_maj$ from $p_k$
9: if $mult > n/2 + f$ then
10: $est_i \leftarrow maj$
11: else $est_i \leftarrow king\_maj$
12: if $k = f + 1$ then $y_i = est_i$
}

}
Message pattern for the phase-king algorithm

\[ p_1, p_2, \ldots, p_k, p_{f+1} \]

\[ \text{round 1, round 2, phase 1, phase f+1} \]
Proof of correctness of the phase-king consensus algorithm

We first show some persistency of agreement:

**Lemma (1)**

*If all non faulty processors prefer \( v \) at the beginning of phase \( k \), then they all prefer \( v \) at the end of phase \( k \), \( 1 \leq k \leq f + 1 \)*

**Proof:**

- If all processors prefer \( v \) at the beginning of phase \( k \), each processor receives at least \( n - f \) copies of \( v \) (including its own value) at the end of the first round of phase \( k \).
- By assumption, \( n \geq 4f \).
- Thus \( n - f \geq n/2 + f \).
- Line 9 implies that they will all set \( \text{est}_i \) to \( v \) by the end of round 2 of phase \( k \).
This immediately implies

- The Validity property
  - since the value at the beginning of the next phase is equal to the value at the end of the current phase

- The termination property
  - since this holds up to phase $f + 1$
Proof of correctness of the phase-king consensus algorithm

- Agreement is achieved by the king breaking ties
- Because there are at most $f$ failures, then at least one coordinator will be correct (i.e., not Byzantine)
- and this coordinator will allow all the correct processors to decide
Proof of correctness of the phase-king consensus algorithm

Lemma (2)

Let \( g \) be a phase whose coordinator is \( p_g \) is nonfaulty. Then all nonfaulty processors finish phase \( g \) with the same preference.

Proof:

1. Suppose that all processors use the coordinator value (Line 11)
   - By assumption, \( p_g \) is nonfaulty, thus it sends the same value to all processors
   - By Lemma (1) they will keep the same value up to the end of phase \( g \)
Proof of correctness of the phase-king consensus algorithm

Lemma (2)

Let \( g \) be a phase whose coordinator is \( p_g \) is nonfaulty. Then all nonfaulty processors finish phase \( g \) with the same preference.

Proof (cont’d):

2. Suppose that some non-faulty processor \( p_i \) uses its own majority value \( v \) (Line 10)
   - Thus \( p_i \) must have received \( n/2 + f \) times value \( v \) in round 1 of phase \( g \)
   - Thus every processor received at least \( n/2 \) times value \( v \) during round 1 pf phase \( g \)
   - Thus all of them set their majority value equal to \( v \)

3. Thus whatever Line 10 or Line 11 is executed, the preferred value is set to \( v \)
Lemma (2)

Let $g$ be a phase whose coordinator is $p_g$ is nonfaulty. Then all nonfaulty processors finish phase $g$ with the same preference.

Proof (cont’d):

1. Thus, at phase $g + 1$ all the processors have the same preference.
2. Lemma (1) implies that they all decide the same value at the end of phase $f + 1$. 
Bibliography

- N. Lynch, Distributed Algorithms, Morgan Kaufmann Publishers