Consensus

Brian Nielsen

bnielsen@cs.aau.dk
Consensus problems

• Examples
  – Mutex: which process is granted access
  – Reliable and ordered Multicast
  – Election
  – Abort/proceed in space shuttle launch
  – Consistent credit/debit bank account

• Fault Tolerance
  – Crash, Omission
  – Byzantine (Arbitrary) failures
  – No message signing
    • Message signing limits the harm a faulty process can do

• Problems
  – Consensus
  – Byzantine generals
  – Interactive consistency
Redundancy

• Components (sensors / memory / processors/processes) may fail
• Critical systems: space / nuclear / train control
• Increase availability ⇒ Dublicate components/functionality

![Diagram of redundancy with components and processes]
Example

• The PASS (Primary Avionics Software System) developed by IBM in 1981, was used in a space shuttle
  – Could have been done on one computer
  – But 4 separate processors were used for fault-tolerance
    • Voting on the outcome
Space Shuttle DS hardware
Radiation

- The Natural (and Hostile) Radiation Environment Poses a Significant Threat to Many Electronic Devices
  - Single Event Upset (SEU), Single Event Latchup (SEL), …

Consensus in a synchronous systems w. crash failures
Communication Model

• Reliable point-to-point communication
• Pairwise channels (complete graph)
• Synchronous system
B-Multicast

Send a message to all processors in one round
Concurrent Multicast

- More processes can multicast at the same round
Concurrent Multicast

$p_2$  a,b

$p_1$  b

$p_5$  a

$p_3$  a,b

$p_4$  a,b
Crash Failures

Faulty processor

Diagram:
- Node $p_1$ is marked as faulty.
- Edges labeled with 'a' connect nodes $p_1$, $p_2$, and $p_3$.
Un-reliable multicast

Faulty processor

B-multicast is unreliable

• Some of the messages are never delivered, if sender crashes
Un-reliable multicast

Faulty processor $p_1$

$p_2$, $p_3$, $p_4$, $p_5$
Crash-failures

Failure

After failure the process disappears from the network
Consensus for three processes

Consensus algorithm

Selection function:
• \(d_i = \text{majority}(v_1, \ldots, v_n)\)
• \(d_i = \text{minimum}(v_1, \ldots, v_n)\)
• ...

\(v_3 = \text{abort}\)

\(P_3 \text{ (crashes)}\)
Consensus

- **Termination:** Eventually each correct process $p_i$ sets its decision variable $d_i$.

- **Agreement:** The decision value of all correct processes is the same: if $p_i$ and $p_j$ are correct and have entered their decided state, then $d_i = d_j$ (for all $i, j \geq 1..N$).

- **Integrity:** If the correct processes all proposed the same value, then any correct process in the decided state has chosen that value.
Consensus

Start

Everybody has an initial proposed value $v_i$
Consensus

Agreement: Everybody decides on the same value: $d_i = d_j$ (for all $i,j \in \{1, \ldots, N\}$)
**Integrity:** If the correct processes all proposed the same value, then any correct process in the decided state has chosen that value.
An Algorithm?

Each process $p_i$:

1. B-multicast its value to all processes
2. Decide on the minimum

(only one round is needed)
An Algorithm?

Start

Diagram:

- Node 0
- Node 1
- Node 2
- Node 3
- Node 4

Connections:
- 0 to 1
- 0 to 2
- 0 to 3
- 0 to 4
- 1 to 2
- 1 to 3
- 1 to 4
- 2 to 3
- 2 to 4
- 3 to 4
An Algorithm?

B-multicast values

0, 1, 2, 3, 4

0, 1, 2, 3, 4

0, 1, 2, 3, 4

0, 1, 2, 3, 4

0, 1, 2, 3, 4

0, 1, 2, 3, 4

0, 1, 2, 3, 4

0, 1, 2, 3, 4
An Algorithm?

Decide on minimum

0, 1, 2, 3, 4
An Algorithm?
An Algorithm?

Without Failures, this algorithm gives consensus

If everybody starts with the same initial value, everybody decides on that value (minimum)
Consensus w. Crash Failures

The simple algorithm doesn’t work

Each process $p_i$:

1. B-multicast value to all processors
2. Decide on the minimum
Consensus w. Crash Failures

Start

Not all processes receives the proposed value from the failed process
Consensus w. Crash Failures

Communicated values

0, 1, 2, 3, 4

fail

1, 2, 3, 4

1

2

3

4

0, 1, 2, 3, 4

0, 1, 2, 3, 4

1, 2, 3, 4
Consensus w. Crash Failures

Decide on minimum fail

0,1,2,3,4

0,1,2,3,4

1,2,3,4

1,2,3,4

1

0

1

0

0,1,2,3,4

1,2,3,4
Consensus w. Crash Failures

Finish

fail 0

0 1

1 0

No Consensus!!!
f-resiliency

- \textit{f-resilient consensus algorithm}
  - Guarantees consensus with up to $f$ failed process
Example 3-resiliency

Example: The input and output of a 3-resilient consensus algorithm

Start

Finish
An f-resilient algorithm

Round 1:
Each process B-multicast its value

Round 2 to round f+1:
B-multicast any new received values

End of round f+1:
Decide on the minimum value received
Consensus in a synchronous system

Algorithm for process $p_i \in g$: algorithm proceeds in $f - 1$ rounds

On initialization

\[ \text{Values}_i^1 := \{v_i\}; \hspace{1cm} \text{Values}_i^0 = \{\}\; . \]

In round $r$ ($1 \leq r \leq f - 1$)

\[ \text{B-multicast}(g, \text{Values}_i^{r-1} - \text{Values}_i^{r-1}); \hspace{1cm} \text{// Send only values that have not been sent} \]

\[ \text{Values}_i^r := \text{Values}_i^{r-1}; \]

\[ \text{while (in round } r) \]

\[ \{ \]

\[ \text{On B-deliver}(V_j) \text{ from some } p_j \]

\[ \text{Values}_i^{r-1} := \text{Values}_i^{r-1} \cup V_j; \]

\[ \} \]

After $(f - 1)$ rounds

\[ \text{Assign } d_i = \text{minimum}(\text{Values}_i^{f-1}); \]

A round is completed in $T$ secs

$\Rightarrow$ synchronous system
Example

Start

0

1

2

3

4

f=1 failures, f+1 = 2 rounds needed
Example: $f=1$

Round 1

0, 1, 2, 3, 4

(new values)

B-multicast all values to everybody
Example: $f=1$

Round 2

0,1,2,3,4

1

4

0,1,2,3,4

2

3

0,1,2,3,4

B-multicast all new values to everybody
Example: \( f=1 \)

Decide on minimum value: for all \( i \): \( d_i=0 \),
Example run 1: f=2

Example: f=2 failures, f+1 = 3 rounds needed
Example run 1: \( f=2 \)

Round 1

1, 2, 3, 4

1, 2, 3, 4

0

0

1, 2, 3, 4

4

0, 1, 2, 3, 4

2

B-multicast all values to everybody
Example run 1: $f=2$

Round 2

Failure 1

B-multicast new values to everybody
Example run 1: $f=2$

Round 3

0,1,2,3,4

1

Failure 1

4

0, 1,2,3,4

B-Multicast new values to everybody

0,1,2,3,4

2

Failure 2

3

0,1,2,3,4
Example run 1: $f=2$

Finish

$0,1,2,3,4$

$0$

Failure 1

$0,1,2,3,4$

$0$

Decide on the minimum value

$0,1,2,3,4$

$0$

Failure 2

$0,1,2,3,4$
Example run 2: $f=2$

Start

0

1

2

3

4
Example run 2: $f=2$

Round 1:

1, 2, 3, 4

1, 2, 3, 4

0

Failure 1

B-multipicast all values to everybody
Example run 2: $f=2$

Round 2

Failure 1

0, 1, 2, 3, 4

B-multicast new values to everybody

Remark: At the end of this round all processes know about all the other values.
Example run 2: \( f=2 \)

Round 3

\( 0,1,2,3,4 \)

Failure 1

1

0,1,2,3,4

4

Failure 2

0,1,2,3,4

B-multicast new values to everybody

(no new values are learned in this round)
Example run 2: $f=2$

Finish

0,1,2,3,4

0

0,1,2,3,4

0

0,1,2,3,4

0

0,1,2,3,4

0

0,1,2,3,4

0

Decide on minimum value
Observation

Example:
5 failures,
6 rounds

If there are \( f \) failures and \( f+1 \) rounds then there is a round with no failed process.
Need for f+1 Rounds

- At the end of the round with no failure:
  - Every (non faulty) process knows about all the values of all other participating processes
  - This knowledge doesn’t change until the end of the algorithm
- Therefore, at the end of the round with no failure:
  everybody would decide the same value
- The exact position of this ‘good’ round is not known:
  - In worst-case we need f+1 rounds
Worst-case Scenario

Round 1

before process $p_i$ fails, it sends its value $a$ to only one process $p_k$
before process $p_k$ fails, it sends value $a$ to only one process $p_m$
Worst-case Scenario

Round 1 2 3 f

At the end of round f only one process $P_n$ knows about value a
Worst-case Scenario

Round 1 2 3 f decide

Process $p_n$ may decide a, and all other processes may decide another value (b)
Worst-case Scenario

Round | 1 | 2 | 3 | f | decide

Therefore f rounds are not enough
At least f+1 rounds are needed
A Lower Bound

• Theorem
  – Any $f$-resilient consensus algorithm requires at least $f+1$ rounds
Byzantine Failures
The Byzantine generals problem

- Turkish invasion into Byzantium
  - Byzantine generals have to agree on attack or retreat
  - The enemy works by corrupting the soldiers
  - Byzantine generals are notoriously treacherous ...
  - The loyal generals have to prevent traitors from spoiling a coordinated attack
  - Messengers are sent to each other camps
  - Orders are distributed by exchange of messages, corrupt soldiers violate protocol at will
  - But corrupt soldiers can’t intercept and modify messages between loyal troops
  - The gong sounds slowly: there is ample time for loyal soldiers to exchange messages (all to all)
Byzantine Failures

Faulty processor

\( v_1 = a \)

• Aka. Arbitrary Faults
  • Different processes receive different values
  • Ommision failures
  • Crash Failure
Byzantine Failures

Round 1
- $p_1$
- $p_2$
- $p_3$
- $p_4$
- $p_5$

Round 2
- $p_1$
- $p_2$
- $p_3$
- $p_4$
- $p_5$

Round 3
- $p_1$
- $p_2$
- $p_3$
- $p_4$
- $p_5$

Round 4
- $p_1$
- $p_2$
- $p_3$
- $p_4$
- $p_5$

Round 5
- $p_1$
- $p_2$
- $p_3$
- $p_4$
- $p_5$

Round 6
- $p_1$
- $p_2$
- $p_3$
- $p_4$
- $p_5$

After failure a byzantine process may continue functioning in the network
Three byzantine generals

\[ p_1 \text{ (Commander)} \]

\[ p_2 \]

\[ p_3 \]

- Commanding general says attack or retreat!
- Processes may fail arbitrarily
- Processes must reach consensus
Byzantine Generals

- **Termination:** Eventually each correct process sets its decision variable.

- **Agreement:** The decision value of all correct process is the same: if $p_i$ and $p_j$ are correct and have entered their decided state, then $d_i = d_j$ (for all $i, j \in 1..N$).

- **Integrity:** If the commander is correct, then all correct processes decide on the value that the commander proposed.
A Theorem

- $N$ processes must tolerate $f$-faults
- *There is no $f$-resilient algorithm if $N \leq 3f$*
- **Outline**
  1. Impossibility with 3 processes case,
  2. Impossibility if $N \leq 3f$
  3. An algorithm for $N \geq 3f+1$ in synchronous systems
  4. Impossibility of consensus in asynchronous systems
Impossibility of Three Byzantine Generals

Notation:
1: v ~ p_1 says 1
2: 1: v ~ p_2 says p_1 says v

Faulty processes are shown shaded

1. Left: p_2 gets conflicting information. Which is correct?
2. If commander is correct p_2 and p_3 must decide v accordingly (integrity)
3. Right: Symmetrically, p_2 must decide w and p_3 must decide x
4. An algorithm cannot distinguish scenarios: No Agreement
Impossibility of $N \leq 3f$ Byzantine Generals

Reduction:
Each process $q$ simulates $N/3$ processes using algorithm $X$
Impossibility of $N \leq 3f$ Byzantine Generals

When a ‘$q$’ fails $n/3$ then processes fail too.
Impossibility of $N \leq 3f$ Byzantine Generals

Finish of algorithm $X$

$\frac{p_{2n}+1}{3} \ldots p_n$

fails

algorithm $X$ tolerates $n/3$ failures
Impossibility of $N \leq 3f$ Byzantine Generals

Final decision

We reached consensus with 1 failure

Previously shown Impossible!!!

algorithm X cannot exist
Four byzantine generals

Faulty processes are shown shaded

\[ p_2 \text{ and } p_4 \text{ agrees: } \]
\[ d_2 = \text{majority } (v, v, u) = v \]
\[ d_4 = \text{majority } (v, v, w) = v \]

\[ p_2, p_3, \text{ and } p_4 \text{ agrees: } \]
\[ d_2 = d_2 = d_4 = \text{majority } (v, u, w) = \perp \]
\[ \Rightarrow \text{Use common default value} \]
Cost of Byzantine Generals

• Requires $f+1$ rounds,
• Sends $O(n^{f+1})$ messages
• If we use digital signatures a solution exist with $O(n^2)$ messages ($f+1$ rounds)
  – False claims not possible:
    – If ”p says v” other processes can detect if ”q says p says w”
• Truely arbitrary failures are rare.
Impossibility of Consensus in asynchronous systems

- No algorithm exists to reach consensus
  - (Consensus may possibly (very often) be reached, but cannot always guaranteed)
  - Neither for crash or byzantine failures
- Eg. Two-army problem:
  - There is some program continuation that avoids consensus
- No guaranteed solution to
  - Byzantine generals problem
  - Interactive consistency
  - Totally ordered reliable multicast
The two-army problem:
1. Sparta and Carthage together can beat Bad guys but not individually. Therefore, they have to decide to attack at exactly the same time.
2. Sparta general sends a message to Carthage general to attack at noon
3. How does he know that Carthage general received the message?

Arbitrarily slow processes (or channels) are indistinguishable from crashed ones (omission)
Workarounds in an asynchronous system

- **Masking faults:**
  - restart crashed process and use persistent storage
  - Eg recovery files like in databases

- **Use failure detectors:**
  - make failure *fail-silent* by discarding messages

- **Probabilistic algorithms:**
  - conceal strategy for adversary
END