Early Consensus in Message-passing System Enriched with a Perfect Failure Detector and its Application in the Theta Model

François Bonnet, Michel Raynal
IRISA / University of Rennes 1

EDCC 2010
Valencia, Spain
Outline

1. Introduction
   - The consensus problem
   - Failure detectors

2. Distributed system model
   - Asynchronous model
   - Perfect failure detector $P$

3. Consensus algorithms
   - Basic algorithm
   - Early-deciding algorithm

4. $P$ in the $\theta$ model
   - The $\theta$ model
   - Algorithm

5. Conclusion
Consensus Problem

Consensus

Each process proposes a value \( v \) and have to decide a value s.t.:

- **Validity**: A decided value has been proposed by some process,
- **Agreement**: No two processes decide different values,
- **Termination**: All correct processes must decide.
Consensus Problem

Consensus

Each process proposes a value $v$ and have to decide a value s.t.:

- **Validity**: A decided value has been proposed by some process,
- **Agreement**: No two processes decide different values,
- **Termination**: All correct processes must decide.

Consensus in synchronous system

- Easily solvable,
- Requires at least $t + 1$ rounds to tolerate $t$ crashes,
Consensus Problem

Consensus

Each process proposes a value $v$ and have to decide a value s.t.:

- **Validity**: A decided value has been proposed by some process,
- **Agreement**: No two processes decide different values,
- **Termination**: All correct processes must decide.

Consensus in synchronous system

- Easily solvable,
- Requires at least $t + 1$ rounds to tolerate $t$ crashes,
- Can be solved in $\min(f + 2, t + 1)$ rounds (optimal result).
Consensus Problem

Consensus

Each process proposes a value $v$ and have to decide a value s.t.:

- **Validity**: A decided value has been proposed by some process,
- **Agreement**: No two processes decide different values,
- **Termination**: All correct processes must must decide.

Consensus in asynchronous system

- Not solvable [well-known impossibility FLP].
How to circumvent the impossibility?

- **Weaken the Problem**
  - Conditions: Restrict the set of possible inputs.
  - Probabilistic termination: Use of randomization.
How to circumvent the impossibility?

Weaken the Problem

- Conditions: Restrict the set of possible inputs.
- Probabilistic termination: Use of randomization.

Strengthen the System

- Use of Failures Detectors (FD):
  - Give additional informations to each process.
Active research in FD

Find new FDs

- Make solvable some unsolvable problems,
- Rank FDs in some hierarchies.
Active research in FD

Find new FDs
- Make solvable some unsolvable problems,
- Rank FDs in some hierarchies.

Optimality
- Find the “weakest” FD for a given problem,
- Find the “best” algorithm for a given FD and problem.
Active research in FD

Find new FDs
- Make solvable some unsolvable problems.
- Rank FDs in some hierarchies.

Optimality
- Find the “weakest” FD for a given problem.
- Find the “best” algorithm for a given FD and problem.
Asynchronous Message-Passing Model

- Set \( \Pi = \{ p_1, p_2, \ldots, p_n \} \) of \( n \) processes,
- Up to \( 0 \leq t < n \) crashes,
- Communication through asynchronous reliable channels;
  - No loss of message,
  - No duplication of message,
  - Finite but not bounded delivery time.
Asynchronous Message-Passing Model

Set $\Pi = \{p_1, p_2, \ldots, p_n\}$ of $n$ processes,

- Up to $0 \leq t < n$ crashes,
- Communication through asynchronous reliable channels;
  - No loss of message,
  - No duplication of message,
  - Finite but not bounded delivery time.

Number of crashes in a given execution

- $t$: the bound on the number of crashes,
- $f$: the actual number of crashes; $0 \leq f \leq t$. 
Perfect failure detector $P$

Some definitions (with an hidden global time)
- $Faulty(\tau)$: set of processes that crashed before $\tau$,
- $Alive(\tau)$: set of processes that did not crashed before $\tau$,
- Correct, Faulty: set of process that never (resp. ever) crashes.
Some definitions (with an hidden global time)

- **Faulty(\(\tau\))**: set of processes that crashed before \(\tau\),
- **Alive(\(\tau\))**: set of processes that did not crashed before \(\tau\),
- **Correct, Faulty**: set of process that never (resp. ever) crashes.

Formal definition of \(P\)

\(P\) outputs to \(p_i\) at time \(\tau\) a set \(\text{suspected}_i^{\tau}\) of process identities such that:

- Completeness.
  \[\exists \tau : \forall \tau' \geq \tau, \forall i \in \text{Correct}, \forall j \in \text{Faulty} : j \in \text{suspected}_i^{\tau'}\]
- Strong accuracy.
  \[\forall \tau : \forall i, j \in \text{Alive}(\tau) : j \notin \text{suspected}_i^{\tau}\]
Our goals

Reach the synchronous bounds in asynchronous system

- Solve the consensus in $t + 1$ rounds with $P$,
- Solve the consensus in $\min(f + 2, t + 1)$ rounds with $P$,
- Implement $P$ in the $\theta$ model.
Comparison with synchronous system

Problem: Bad phenomenon due to asynchrony

- $i \in \text{suspected}_j$
- $i \not\in \text{suspected}_j$ (forever)

Solution: Eliminate the variations of $P$

Each process $p_i$ maintains a set $\text{crashed}_i$ such that:

$\text{crashed}_i \leftarrow \text{crashed}_i \cup \text{suspected}_i$.

F. Bonnet, and M. Raynal
Comparison with synchronous system

Problem: Bad phenomenon due to asynchrony

Solution: Eliminate the variations of $P$

Each process $p_i$ maintains a set $crashed_i$ such that:

- $crashed_i$ is a monotonous version of $P$:
  
  $$crashed_i \leftarrow crashed_i \cup suspected_i.$$
Basic algorithm with $P$

**operation** propose ($v_i$):

1. $est_i \leftarrow v_i$; $r_i \leftarrow 1$;
2. **while** $r_i \leq t + 1$ do
   3. **begin** asynchronous round
   4. broadcast $EST (r_i, est_i)$;
   5. **wait until** ($\forall j \notin crashed_i$: ($EST (r_i, -)$ received from $p_j$));
   6. let $rec_from_i = \{1, \ldots, n\} \setminus crashed_i$;
   7. let $est_rec_i = \{est$ from the processes in $rec_from_i\}$;
   8. $est_i \leftarrow \min(est_rec_i)$;
   9. $r_i \leftarrow r_i + 1$
10. **end** asynchronous round
11. **end** while;
12. return ($est_i$).
Basic algorithm with $P$

**operation** propose ($v_i$):

1. $est_i \leftarrow v_i$; $r_i \leftarrow 1$;
2. while $r_i \leq t + 1$ do
   3. begin asynchronous round
   4. broadcast $est$ ($r_i, est_i$);
   5. wait until ($\forall j \notin crashed_i$: $(est \ (r_i, \_)$ received from $p_j)$;
   6. let $rec_{from_i} = \{1, \ldots, n\} \setminus crashed_i$;
   7. let $est_{rec_i} = \{est \ from \ the \ processes \ in \ rec_{from_i}\}$;
   8. $est_i \leftarrow \min(est_{rec_i})$;
   9. $r_i \leftarrow r_i + 1$
10. end asynchronous round
11. end while;
12. return ($est_i$).

**Efficiency**

Always decide in $t + 1$ rounds.
Problem of the previous algorithm

It always requires $t + 1$ rounds even if there are few crashes (or no crash at all).
Early-decision

Problem of the previous algorithm

It always requires \( t + 1 \) rounds even if there are few crashes (or no crash at all).

Solution

Adapt the number of rounds to the real number of crashes \( f \).
Implementation of early-decision

**Conditions to decide/stop**

A process decides and stops iff it knows that:

- it has the smallest estimate in the system,
- at least one correct process knows that value.
Implementation of early-decision

### Conditions to decide/stop

A process decides and stops iff it knows that:
- it has the smallest estimate in the system,
- at least one correct process knows that value.

### Implementation

Each process $p_i$ manages two additional variables
- $i\_know_i$: boolean that indicates if it has the smallest estimate,
- $they\_know_i$: set of processes that (from $p_i$’s point of view) have the smallest estimate,
Early-deciding algorithm with $P$

**operation** propose ($v_i$):

1. $est_i \leftarrow v_i$; $r_i \leftarrow 1$; $they\_know_i \leftarrow \emptyset$; $i\_know_i \leftarrow false$;
2. while $r_i \leq t + 1$ do
3. begin asynchronous round
4. broadcast $est (r_i, est_i, i\_know_i)$;
5. wait until $\forall j \notin ((crashed \cup they\_know_i) \setminus \{i\})$: $(est (r_i, -, -))$ received from $p_j$;
(N1) let crashed_or_knowing$_i$ be the set $(crashed \cup they\_know_i)$ when the wait terminates;
6. let rec_from$_i$ = $\{1, \ldots, n\} \setminus$ crashed_or_knowing$_i$;
7. let est_rec$_i$ = $\{est$ received during $r_i$ from the processes in rec_from$_i\}$;
8. $est_i \leftarrow \min(\text{est_rec}_i)$;
(N2) $they\_know_i \leftarrow they\_know_i \cup \{x \mid est (r_i, -, true) \text{ rec. from } p_x \text{ with } x \in \text{rec_from}_i\}$;
(N3) if $(|crashed \cup they\_know_i| \geq t + 1) \land i\_know_i$ then return $(est_i)$ end if;
(N4) let some_known$_i = (\exists est (r_i, -, true) \text{ received from } p_x \text{ with } x \in \text{rec_from}_i)$;
(N5) $i\_know_i \leftarrow (some\_known_i) \lor (|\text{rec_from}_i| \geq n - r_i + 1)$;
9. $r_i \leftarrow r_i + 1$
10. end asynchronous round
11. end while;
12. return $(est_i)$.
Early-deciding algorithm with $P$

**operation** propose ($v_i$):

1. $est_i \leftarrow v_i$; $r_i \leftarrow 1$; $they\_know_i \leftarrow \emptyset$; $i\_know_i \leftarrow false$;
2. while $r_i \leq t + 1$ do
   3. begin asynchronous round
   4. broadcast $EST\ (r_i, est_i, i\_know_i)$;
   5. wait until $(\forall j \notin ((crashed_i \cup they\_know_i) \setminus \{i\}) : \ (EST\ (r_i, \_\_\_, \_\_\_) \text{ received from } p_j)$;
   6. let $crashed\_or\_knowing_i$ be the set $(crashed_i \cup they\_know_i)$ when the wait terminates;
   7. let $est\_rec_i = \{ est \text{ received during } r_i \text{ from the processes in } rec\_from_i \}$;
   8. $est_i \leftarrow \min(est\_rec_i)$;
   9. $they\_know_i \leftarrow they\_know_i \cup \{x | \ (EST\ (r_i, \_\_\_, true) \text{ rec. from } p_x \text{ with } x \in rec\_from_i \}$;
   10. if $(|crashed_i \cup they\_know_i| \geq t + 1) \land i\_know_i)$ then return ($est_i$) end if;
   11. let $some\_knows_i = (\exists \ (EST\ (r_i, \_\_\_, true) \text{ received from } p_x \text{ with } x \in rec\_from_i)$;
   12. $i\_know_i \leftarrow (some\_knows_i) \lor (|rec\_from_i| \geq n - r_i + 1)$;
   13. $r_i \leftarrow r_i + 1$
14. end asynchronous round
15. end while;
16. return ($est_i$).

**Efficiency**

Decide in $\min(f + 2, t + 1)$ rounds.
The asynchronous $\theta$ model (informally)

Bound $\theta$ on transmission delays

- $\delta^+$ denotes the maximal transit time for a message,
- $\delta^-$ denotes the minimal transit time for a message,
- $\theta \geq \left\lceil \frac{\delta^+}{\delta^-} \right\rceil$. 
The asynchronous $\theta$ model (informally)

**Bound $\theta$ on transmission delays**

- $\delta^+$ denotes the maximal transit time for a message,
- $\delta^-$ denotes the minimal transit time for a message,
- $\theta \geq \lceil \frac{\delta^+}{\delta^-} \rceil$.

**Remarks**

- Only $\theta$ is known by processes,
- The system does not refer directly to real time.
Ping/Pong

Key idea to build $P$ (example with $\theta = 3$)

$3(2 \times \delta^-)$

$2 \times \delta^+$
Building $P$ in the $\theta$ model

```
init suspected_i ← \emptyset;
    for each j \neq i do send PING () to p_j end for.

when PING () is received from p_j: send PONG () to p_j.

when PONG () is received from p_j:
    for each k \neq j do
        if (k \notin suspected_i) then
            count_i[j, k] ← count_i[j, k] + 1;
            if (count_i[j, k] > \theta) then
                suspected_i ← suspected_i \cup \{k\}
            else
                count_i[k, j] ← 0
            end if
        end if
    end for;
    send PING () to p_j.
```
Contributions

- Early-deciding algorithm in asynchronous system enriched with $P$,
- Implementation of $P$ in the $\theta$ asynchronous model.

Open problems

- Find easier algo/proof for early-deciding consensus,
- Prove formally the optimality.