Chapter 10

Consensus and Interactive Consistency in Synchronous Systems Prone to Process Crash Failures

This first chapter on agreement in synchronous systems focuses on the consensus and interactive consistency (also called vector consensus) agreement abstractions. It first defines these abstractions, and presents algorithms that build them in the presence of any number of process crashes in the system model $CSMP_{n,t}[\emptyset]$. All these algorithms are round-based (as defined in the system model). The chapter also shows that $(t+1)$ is a lower bound on the number of rounds for any algorithm implementing these abstractions in the system model $CSMP_{n,t}[\emptyset]$.

Keywords Agreement, Binary vs multivalued, Atomic crash, Atomic round, Consensus, Convergence, Hamming distance, Interactive consistency, Lower bound, Process crash failure, Round-based algorithm, Uniformity, Valence, Vector consensus, Synchronous system.

10.1 Consensus in the Crash Failure Model

10.1.1 Definition

Consensus in the process crash failure model The consensus problem is one of the most celebrated problems of fault-tolerant distributed computing. It abstracts a lot of problems where – in one way or another – processes have to agree. This problem can be captured by a distributed object, i.e., a distributed agreement abstraction defined as follows.

The consensus abstraction provides the processes with a single operation denoted propose() which takes a value as an input parameter, and returns a value. If a process p_i invokes propose(v_i) and obtains the value w, we say "p_i proposes v_i", and "p_i decides w". This agreement abstraction is defined by the following properties, where CC stands for consensus in the crash failure model. The definition is the same for both the synchronous model $CSMP_{n,t}[\emptyset]$ and the asynchronous model $CAMP_{n,t}[\emptyset]$. It is assumed that all processes invoke the operation propose() (hence, this is a one-shot operation).

- CC-validity. A decided value is a proposed value.
- CC-agreement. No two processes decide different values.
- CC-termination. Each correct process decides a value.

The CC-validity and CC-agreement properties define the safety property of consensus. CC-validity relates the outputs to the inputs (the output is not a predefined value, which would make the problem trivial and not application-relevant), and CC-agreement defines the quality of the output (there is a

M. Raynal, *Fault-Tolerant Message-Passing Distributed Systems*,
https://doi.org/10.1007/978-3-319-94141-7_10

single decided value). CC-termination is a liveness property stating that the invocation of propose() by a correct process always terminates.

Consensus objects are one-shot objects. This means that, if $CONS$ is a consensus object, a process invokes $CONS$.propose() once (if it does not crash before the invocation).

Consensus as an input vector/output vector relation Fig. 1.5 (Section 1.3) has shown that some distributed computing problems can be captured as an input/output relation on vectors of size n, where the input vector I is such that $I[i]$ represents the input of p_i, and the output vector O is such that $O[i]$ represents the output of p_i.

The consensus abstraction can be expressed in terms of such a relation. An input vector I is the vector containing the values proposed by the processes. Given an input vector I, several output vectors O are possible. Those are the vectors containing the same value v in all their entries, where v is any value present in the input vector I.

Uniform vs non-uniform consensus The previous definition is sometimes called *uniform consensus*, in the sense that it does prevent a process that decides and then crashes from deciding differently from the correct processes. A weaker version of the problem, called *non-uniform consensus*, allows a process that crashes to decide differently from the other processes. It is defined by the same CC-validity and CC-termination properties plus the following weaker agreement property.

- Non-uniform CC-agreement. No two correct processes decide different values.

More generally, when considering the process crash failure model, a *uniform* property directs any process that crashes to behave as a correct process (before crashing). In the case of consensus, it is not because a process crashes after having decided that it is allowed to decide a value different from the one decided by the correct processes.

In the following, except when explicitly indicated, we always consider uniform properties.

Lower bound As we will see, consensus can be solved in the synchronous crash failure model for any value $t < n$, i.e., in the unconstrained system model $CSMP_{n,t}[\emptyset]$.

Binary vs multivalued consensus Let $\mathcal{V}$ be the set of values that can be proposed to a consensus object. If $|\mathcal{V}| = 2$, the consensus is binary. In this case, it is usually considered that $\mathcal{V} = \{0, 1\}$. If $|\mathcal{V}| > 2$, the consensus is multivalued. In this case, the set $\mathcal{V}$ can be finite or infinite.

10.1.2 A Simple (Unfair) Consensus Algorithm

A simple consensus algorithm A process p_i invokes the operation propose (v_i) where v_i is the value it proposes. It terminates when it executes the statement return(v) and v is then the value it decides.

The principle of the algorithm is pretty simple. As at most t processes may crash (model assumption), any set of $(t+1)$ processes contains at least one correct process. (If more than t processes crash, we are outside the model. In that case there is no guarantee. More generally, if an algorithm is used in a more severe failure model than the one it is intended for, it is allowed to behave arbitrarily.) It follows that taking any set of $(t + 1)$ processes we can always rely on one of them to ensure that a single value is decided.

The corresponding algorithm is described in Fig. 10.1. Each process manages a local variable est_i that contains its estimate of the decision value; est_i is consequently initialized to v_i (line 1). Then, the processes execute synchronously $(t + 1)$ rounds (line 2), each round being coordinated by a process, namely, round r is coordinated by process p_r. The coordinator of round r broadcasts its current estimate (message EST(), line 4). Let us notice that, as a round is coordinated by a single process, there is at most one value broadcast per round. During a round, a process p_i updates its estimates est_i

if it receives the current estimate of the current round coordinator (line 5). Finally, at the end of the last round, p_i decides (returns) the current value of its estimate est_i.

```
operation propose (v_i) is
(1) est_i ← v_i;
(2) when r = 1, 2, ..., (t + 1) do
(3) begin synchronous round
(4)     if (i = r) then broadcast EST(est_i) end if;
(5)     if (EST(v) received during round r) then est_i ← v end if;
(6)     if (r = t + 1) then return(est_i) end if
(7) end synchronous round.
```

Figure 10.1: A simple (unfair) t-resilient consensus algorithm in $CSMP_{n,t}[\emptyset]$ (code for p_i)

Theorem 39. *Let $1 \leq t < n$. The algorithm described in Fig. 10.1 solves the consensus problem in the system model $CSMP_{n,t}[\emptyset]$.*

Proof The CC-validity property (a decided value is a proposed value) is trivial. The CC-termination property (every correct process decides) is an immediate consequence of the synchrony assumption: the system automatically progresses from one round to the next one (with the guarantee that the messages sent in a round are received in the very same round).

The CC-agreement property (no two processes decide differently) is an immediate consequence of the following observation. Due to the assumption on the maximum number t of processes that may crash, there is at least one round that is coordinated by a correct process. Let p_c be such a process. When $r = c$, p_c sends its current estimate $est_c = v$ to all the processes, and any process p_j that has not crashed updates est_j to v. It follows that all the processes that have not crashed by the end of round r have their estimates equal to v, and consequently no other value can be decided. $\square_{Theorem\ 39}$

Time and message complexities The algorithm requires $(t + 1)$ rounds. Moreover, at most one message is broadcast at each round, i.e., $(n-1)$ messages. Let b be the bit size of the proposed values. The bit complexity is consequently $(n - 1)(t + 1)b$.

Unfairness with respect to proposed values While correct, the previous algorithm has the following "drawback": for any $j \in \{(t + 1), \ldots, n\}$, there is no run in which the value v_j proposed by p_j can be decided (if v_j is not a value proposed by a coordinator process). In that sense, the algorithm is unfair.

This unfairness can be eliminated by adding a preliminary shuffle round ($r = 0$) during which the processes exchange their values. This is done by inserting the statements "broadcast EST(est_i); $est_i \leftarrow$ any estimate value received" between line 1 and line 2. This makes the algorithm fair, but is obtained at the additional cost of one round.

10.1.3 A Simple (Fair) Consensus Algorithm

Let us remember that the input vector of a given a run is the size n vector such that, for any j, its j-th entry contains the value proposed by p_j. No process p_i initially knows this vector, it only knows the value it proposes to that consensus instance.

Principle of the algorithm The idea is for a process to decide, during the last round, a value according to a deterministic rule among all the values it has seen. An example of a deterministic rule is to select the smallest value. This is the rule we consider here. This value is kept in the local variable est_i (initialized to v_i, the value proposed by p_i).

Let us observe that, if a process p_i does not crash and proposes the smallest input value, that value will be decided whatever the values proposed by the other processes. Hence, for any process p_i, there are (a) input vectors in which no two processes propose the same value, and (b) failure patterns, such that the value proposed by p_i is decided in the current run. The algorithm is fair in that sense.

The algorithm is described in Fig. 10.2. The processes execute $(t+1)$ synchronous rounds (line 2). The idea is for a process p_i to broadcast the smallest estimate value it has ever received during each round. But a simple observation shows that this is required only if its estimate became smaller during the previous round (line 4). To this end, p_i manages a local variable denoted $prev\_est_i$ that contains the smallest value it has previously sent (line 6). This variable is initialized to the default value $\perp$ (a value that cannot be proposed to the consensus by the processes).

During a round r, the set $recval_i$ contains the estimate values received by p_i during the current round r (line 5). Due to the synchrony assumption, it contains all estimate values sent to p_i during this round. Before proceeding to the next round, p_i updates est_i (line 7). If r is the last round ($r = t + 1$), p_i decides by invoking return(est_i) (line 8).

```
operation propose (v_i) is
(1) est_i ← v_i; prev_est_i ← ⊥;
(2) when r = 1, 2, ..., (t + 1) do
(3) begin synchronous round
(4)     if (est_i ≠ prev_est_i) then broadcast EST(est_i) end if;
(5)     let recval_i = {values received during round r};
(6)     prev_est_i ← est_i;
(7)     est_i ← min(recval_i ∪ {est_i});
(8)     if (r = t + 1) then return(est_i) end if
(9) end synchronous round.
```

Figure 10.2: A simple (fair) t-resilient consensus algorithm in $CSMP_{n,t}[\emptyset]$ (code for p_i)

Theorem 40. *Let $1 \leq t < n$. The algorithm described in Fig. 10.2 solves the* consensus *agreement abstraction in the system model $CSMP_{n,t}[\emptyset]$.*

Proof As in the previous algorithm, the CC-validity and CC-termination properties are trivial. Hence, we consider only the CC-agreement property.

If a single process decides (we have then $t = n - 1$ and t processes crash), the agreement property is trivially satisfied. Hence, let us suppose that at least two processes p_i and p_j decide. Moreover, let us assume that p_i decides v, and p_j decides v'. We show that $v = v'$. Assuming process p_x has not crashed by the end of round r, let est_x^r denote the value of est_x at the the end of round r.

As both p_i and p_j decide, both execute $t + 1$ rounds. Let us consider p_i. It "learns" (receives for the first time) the value v at some round r (with $r = 0$ if $v = v_i$ the value proposed by p_i itself). As p_i decides $v = est_i^r$ and est_i cannot increase, we have $est_i^r = \cdots = est_i^{t+1}$. There are two cases.

- Case 1: $r < t + 1$ (r is not the last round, and consequently $r + 1$ does exist). In this case, p_i broadcast EST(v) during round $r + 1 \leq t + 1$. As p_j executes the round $r + 1$, it receives v and we have $est_j^{r+1} \leq v$. As est_j never increases, we have $est_j^{t+1} \leq v$.
- Case 2: $r = t+1$. In this case, p_i learns v at round $t+1$ and there are no more rounds to forward v to the other processes. As (a) a process broadcasts a value v at most once, and (b) p_i receives v for the first time at round $(t + 1)$, it follows that v has been forwarded (broadcast) along a chain of $(t+1)$ distinct processes. Due to the model assumption, at least one of these $(t+1)$ processes (say p_x) is correct. As it is correct, p_x broadcast EST(v) during a round r, $1 \leq r \leq t + 1$. (Let us also observe that we necessarily have $r = t + 1$, otherwise p_i would have received EST(v) before the last round.) This is depicted on Fig. 10.3 where $t = 3$, each arrow is associated with a message EST(v), and a cross indicates the crash of the corresponding process. It follows that all processes that execute round r are such that $est_j^r \leq v$, and consequently $est_j^{t+1} \leq v$.

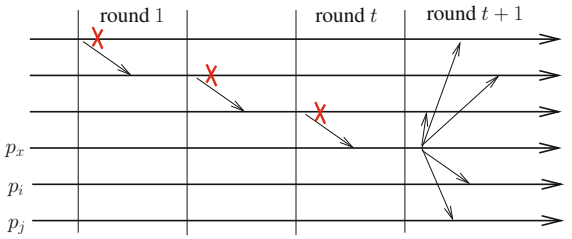

Figure 10.3: The second case of the agreement property (with $t = 3$ crashes)

As $v = est_i^{t+1}$, it follows that we have $est_j^{t+1} \leq est_i^{t+1}$. A symmetry argument where p_i and p_j are exchanged allows us to conclude that $est_j^{t+1} \leq est_i^{t+1}$. Hence, $est_j^{t+1} = est_i^{t+1}$, which concludes the proof of the theorem. $\square_{Theorem\ 40}$

Time and message complexities As with the previous algorithm, this algorithm requires $(t + 1)$ rounds.

During a round, a process send at most $(n - 1)$ messages (we do not count the message it sends to itself), and each message is made up of b bits. Moreover, due to the fact that a process sends an estimate value only if it is smaller than the previous one, a process issues at most $\min(t + 1, |\mathcal{V}|)$ broadcasts, where $\mathcal{V}$ is the set of values that are proposed. It follows that the bit complexity of the algorithm is upper bounded by $n(n - 1)b \times \min(t + 1, |\mathcal{V}|)$.

Interestingly, in the case of binary consensus we have $b = 1$ and $|\mathcal{V}| = 2$. The bit complexity is then $2n(n - 1)$.

10.2 Interactive Consistency (Vector Consensus)

While consensus is an agreement abstraction on a value proposed by the processes, *interactive consistency* is agreement abstraction where the processes agree on the input vector of the proposed values. This is why it is sometimes named *vector consensus*.

10.2.1 Definition

Similar to consensus each process proposes a value. As just indicated, the processes now have to agree on the vector of proposed values. A process can crash before or while it is executing the algorithm. In this case, its entry in the decided vector can be $\perp$. More precisely, interactive consistency in the crash failure model (ICC) is defined by the following properties.

- ICC-validity. Let $D_i[1..n]$ be the vector decided by a process p_i. $\forall\ j \in [1..n] :\ D_i[j] \in \{v_j, \perp\}$ where v_j is the value proposed by p_j. Moreover, $D_i[j] = v_j$ if p_j is correct.
- ICC-agreement. No two processes decide different vectors.
- ICC-termination. Every correct process decides on a vector.

Let us notice that, if $D_i[j] = \perp$ and p_i is correct, it knows that p_j crashed. Whereas, if $D_i[j] \neq \perp$, p_i cannot conclude that p_j is correct.

It is easy to see solve consensus from interactive consistency. As all the processes that decide obtain the same vector, they can use the same deterministic rule to select one of its non-$\perp$ values. However, interactive consistency cannot be solved from consensus. This is because, the value decided by a consensus instance is the value proposed by *any* process. It follows that, in the system model

$CSMP_{n,t}[\emptyset]$, interactive consistency is a stronger (from a commputability point of view) abstraction than consensus.

10.2.2 A Simple Example of Use: Build Atomic Rounds

Atomic round: definition The crash of a process p_i during a round r is *atomic* if the message that p_i is assumed to broadcast during this round is received by none or all its (non-crashed) destination processes. If during a round, all crashes are atomic, the round is an *atomic* round.

Let the synchronous *atomic round-based* model be the basic model $CSMP_{n,t}[\emptyset]$, in which:

- each process broadcasts a message at every round, and
- all crashes are atomic (i.e., all rounds are atomic).

Such a synchronous model simplifies drastically the design of distributed synchronous algorithms. This is because it follows from the previous behavioral properties that all the processes that terminate a round r received exactly the same messages during every round r', $1 \leq r' \leq r$.

From interactive consistency to the atomic round-based model It is consequently worth designing an algorithm that simulates the atomic round-based model on top of the base synchronous model $CSMP_{n,t}[\emptyset]$. Among its many applications, this is exactly what is done by interactive consistency.

The simulation is as follows. Assuming that each process p_i broadcasts a message during each round, let us call ρ the rounds in the atomic round-based model. Considering any round ρ, let m_i^ρ be the message broadcast by p_i during this round of the atomic round-based model. The send and receive phases of such a round ρ are implemented by an interactive consistency instance where m_i^ρ is the value proposed by process p_i to this instance. It follows from its specification that all the processes that terminate the interactive consistency instance associated with round ρ of the atomic round-based model, obtain the very same vector $D[1..n]$, such that $D[j] \in \{m_j^\rho, \perp\}$ and is m_j^ρ if p_j has not crashed by the end of this interactive consistency instance. Hence, as we are about to see, each round ρ of the atomic round-based model can be implemented with $(t+1)$ rounds of the underlying synchronous round-based model $CSMP_{n,t}[\emptyset]$.

10.2.3 An Interactive Consistency Algorithm

Principle of the algorithm The interactive consistency algorithm presented in Fig. 10.4 is based on the same principle as the consensus algorithm described in Fig. 10.2, namely, at every round, each process broadcasts what it learned during the previous round, which is now a set of pairs $\langle$ process id, proposed value$\rangle$.

Given a process p_i, the local variable $view_i$ represents its current knowledge of the values proposed by the other processes, more precisely, $view_i[k] = v$ means that p_i knows that p_k proposed value v, while $view_i[k] = \perp$ means that p_i does not know the value proposed by p_k. Initially, $view_i$ contains only $\perp$s, but its ith entry contains v_i (line 1).

In order to forward the value of a process only once, the algorithm uses pairs $\langle k, v\rangle$ to denote that "p_k proposed value v". The local variable new_i is a (possibly empty) set of such pairs $\langle k, v\rangle$. At the end of a round r, new_i contains the new pairs that p_i learned during this round (lines 9-13). Hence, initially $new_i = \{\langle i, v_i\rangle\}$ (line 1).

- Send phase (line 4). The behavior of a process p_i is simple. When it starts a new round r, p_i broadcasts EST(new_i), if $new_i \neq \emptyset$, to inform the other processes of the pairs it has learned during the previous round.
- Receive phase (lines 5-7). Then, p_i receives round r messages and saves their values in the local array $recfrom_i[1..n]$. Let us observe that it is possible that a process receives no message at some rounds.)

- Local computation phase (lines 8-14). After having reset new_i, p_i updates its array $view_i$ according to the pairs it has received. Moreover, if p_i learns (i.e., receives for the first time) a pair $\langle k, v\rangle$ during the current round, it adds it to the set new_i. Finally, if r is the last round, p_i returns $view_i$ as the vector it decides on.

```
operation propose (v_i) is
(1)  view_i ← [⊥, ..., ⊥]; view_i[i] ← v_i; new_i ← {⟨i, v_i⟩};
(2)  when r = 1, 2, ..., (t + 1) do
(3)  begin synchronous round
(4)      if (new_i ≠ ∅) then broadcast EST(new_i) end if;
(5)      for each j ∈ {1, ..., n} \ {i} do
(6)          if (new_j received from p_j) then recfrom_i[j] ← new_j else recfrom_i[j] ← ∅ end if;
(7)      end for;
(8)      new_i ← ∅;
(9)      for each j such that (j ≠ i) ∧ (recfrom_i[j] ≠ ∅) do
(10)         for each ⟨k, v⟩ ∈ recfrom_i[j] do
(11)             if (view_i[k] = ⊥) then view_i[k] ← v; new_i ← new_i ∪ {⟨k, v⟩} end if
(12)         end for
(13)     end for;
(14)     if (r = t + 1) then return(view_i) end if
(15) end synchronous round.
```

Figure 10.4: A t-resilient interactive consistency algorithm in $CSMP_{n,t}[\emptyset]$ (code for p_i)

10.2.4 Proof of the Algorithm

It would be possible to prove that the previous algorithm satisfies the ICC-agreement property using the same reasoning as in the proof of Theorem 40, i.e., considering the case where a process learns a pair $\langle k, v\rangle$ for the first time during the last round or a previous round. A different proof is given here. This proof is an immediate consequence of Lemma 38 that follows.

The interest of this lemma lies in the fact that it captures a fundamental property associated with the round-based synchronous model where, during each round r, each process (that has not crashed) forwards the values that it has learned during round $r-1$ (if any). The lemma captures the intuition that the "distance" separating the local views of the input vector (as perceived by each process) decreases as rounds progress. To this end, given two vectors $view_i$ and $view_j$, let $\mathsf{dist}(view_i, view_j)$ denote the Hamming distance separating these vectors, namely, $\mathsf{dist}(view_i, view_j) = |\{x \text{ such that } view_i[x] \neq view_j[x]\}|$ (number of entries where the vectors differ).

Lemma 38. *Let $1 \leq t < n$, $1 \leq r < t+1$, p_i and p_j be two processes not crashed at the end of round r, and $view_i^r$ and $view_j^r$ the value of $view_i$ and $view_j$ at the end of round r. We have $\mathsf{dist}(view_i^r, view_j^r) \leq t - (r-1)$.*

Proof Let $\delta(r)$ be the maximal Hamming distance between the vectors of any two processes not crashed by the end of round r. We have to show that $\delta(r) \leq t - (r-1)$.

Claim C. Let r be a failure-free round, and p_i and p_j any two processes that have not crashed by the end of round r. We have $\delta(r') = 0$ for $r \leq r' \leq t+1$.
Proof of the claim. Let us first observe that, at each round r'' such that $1 \leq r'' \leq r$, both p_i and p_j send to the other every new value it has learned during the round $r''-1$ (Observation $O1$). Moreover, as no process crashes during round r, p_i and p_j have received the same set of messages during that round (Observation $O2$). It follows from $O1$ and $O2$ that $view_i$ and $view_j$ are equal at the end of round r. As p_i and p_j are any pair of processes that terminate round r, it follows that $\delta(r) = 0$. Moreover, as from round r no process can learn new values, we trivially have $\delta(r') = 0$ for $r \leq r' \leq t+1$. End of

proof of the claim.

The proof of the lemma considers the failure pattern in the worst case scenario in which t processes crash. Let $c \geq 1$ be the number of processes that have crashed by the end of the first round. The worst situation is when, at the end of the first round, a process p_i has received all the proposed values (i.e., $view_i$ contains only non-$\perp$ values), while another process p_j has received only $n - c$ proposed values (i.e., $view_j$ has c entries equal to $\perp$). It follows that $\delta(1) \leq c$. From then on, no two vectors can differ in more than c entries, and consequently we have $\delta(r) \leq c$ for $1 \leq r \leq t + 1$. The rest of the proof is a case analysis, according to the value of r.

- The first case considers the rounds $1 \leq r \leq t + 1 - c$.
  As $r \leq t + 1 - c \equiv c \leq t - (r - 1)$, it follows from $\delta(r) \leq c$ for $1 \leq r \leq t + 1$, that $\delta(r) \leq c \leq t - (r - 1)$ for the rounds $1 \leq r \leq t + 1 - c$, which proves the lemma for these rounds.
- The second case considers the remaining rounds $t + 1 - c < r \leq t + 1$.
  By the end of the first round, c processes have crashed. The worst case scenario for the next rounds r, $1 \leq r \leq t + 1 - c$, is when there is a crash per round. Otherwise, due to Claim C, we would have $\delta(r') = 0$ from the first round r', $1 \leq r' \leq t+1-c$, during which there is no crash.

Round number r	1	2	...	r'	...	$t + 1 - c$
Number of crashes during r	c	1	...	1	...	1
Total number of crashes	c	$c + 1$	...	$c + (r' - 1)$	...	t

Table 10.1: Crash pattern

In this worst case, we can conclude that there are no more crashes after the round $t + 1 - c$. This is because there are at most t crashes, c before the end of the first round and then one crash per round from round $r = 2$ until round $r = t + 1 - c$. This is depicted in Table 10.1. It then follows from Claim C that $\delta(r') = 0$ for $t + 1 - c < r' \leq t + 1$, which concludes the proof of the lemma.

$\square_{Lemma\ 38}$

Theorem 41. *Let $1 \leq t < n$. The algorithm described in Fig. 10.4 implements the* interactive consistency *agreement abstraction in the system model $CSMP_{n,t}[\emptyset]$.*

Proof The ICC-termination property follows directly from the message synchrony assumption of the synchronous model: if a process does not crash, it necessarily progresses until round $t + 1$. The ICC-agreement property follows from Lemma 38: at round $t = t+1$ we have $dist(view_i^{t+1}, view_j^{t+1}) = 0$.

The ICC-validity property states that the vector $view_i[1..n]$ decided by a process p_i is such that (a) $view_i[k] \in \{v_k, \perp\}$ where v_k is the value proposed by p_i, and (b) $view_i[k] = v_k$ if p_k is correct. Let us assume that p_k is correct. It follows from the algorithm that p_k broadcasts EST($\{\langle k, v_k\rangle\}$) during the first round. Due to the synchrony assumption and the reliability of the communication channels, process p_i receives this message (line 6). Then p_i updates accordingly $view_i[k]$ to v_k (line 11). Finally, let us observe that, due to the test of line 11, any entry $view_i[x]$ is set at most once. Consequently $view_i[k]$ remains forever equal to v_k, which concludes the proof of the validity property. $\square_{Theorem\ 41}$

Time and message complexities As for the previous algorithms, this algorithm requires $(t + 1)$ rounds. A pair $\langle k, v\rangle$ requires $b + \log_2 n$ bits (where b is the number of bits needed to encode a proposed value). As a process broadcasts a given pair $\langle k, v\rangle$ at most once, the bit complexity of the algorithm is upper bounded by $n^2(n - 1)(b + \log_2 n)$ bits (assuming a process does not physically send messages to itself).

From interactive consistency to consensus Consensus can easily be solved as soon as one has an algorithm solving interactive consistency. As the processes that decide in the interactive consistency agreement abstraction decide the very same vector, they can use the same deterministic rule to extract a non-$\perp$ value from this vector (e.g., the first non-$\perp$ value or the greatest value, etc.). The only important point is that they all use the same deterministic rule.

10.2.5 A Convergence Point of View

This section gives another view on the way the algorithm works. Let $VIEW^r[1..n]$ be the vector of proposed values collectively known by the set of processes that terminate round r. More explicitly, $VIEW^r[i] = v_i$ (the value proposed by p_i) if $\exists\ k$ such that $view^r_k[i] = v_i$, otherwise $VIEW^r[i] = \perp$. This means that $VIEW^r[1..n]$ is the "union" of the local vectors $view_k[1..n]$ of the processes p_k that terminate round r. This vector represents the knowledge on "which processes have proposed which values" that an external omniscient observer could have, which would see inside all processes that terminate round r.

Definition $(V1 \leq V2) \stackrel{def}{=} \forall x \in [1..n] : (V1[x] \neq \perp) \Rightarrow (V1[x] = V2[x])$.

The algorithm satisfies the following properties.

Property 1. $\forall r \in [0..t] : VIEW^{r+1} \leq VIEW^r$.

This property follows from the fact that crashes are stable (once crashed, a process never recovers). It states that global knowledge cannot increase.

Property 2. $\forall i \in [1..n] : \forall r \in [1..t+1] : view^r_i \leq view^{r+1}_i$.

This property follows from the fact that no value is ever withdrawn by a process p_i from its local array $view_i$. It states that local knowledge of a process can never decrease.

Property 3. $\forall i \in [1..n] : \forall r \in [1..t+1] : view^r_i \leq VIEW^r$.

This property states that, at the end of any round r, a process cannot know more than what is known by the whole set of processes still alive at the end of the round.

The interactive consistency algorithm, based on the fact that global knowledge cannot increase and local knowledge cannot decrease, is a distributed algorithm that directs the processes to converge to the same vector $VIEW^{t+1}$.

10.3 Lower Bound on the Number of Rounds

This section shows that, when considering the synchronous crash-prone model $CSMP_{n,t}[\emptyset]$, any round-based consensus algorithm that copes with t process crashes requires at least $(t+1)$ rounds. This means that there is no algorithm that always solves consensus in at most t rounds ("always" means "whatever the failure pattern, defined as the subset of processes that crash and the time instants at which they crash").

As any algorithm that implements the interactive consistency agreement abstraction can be used to solve consensus, it follows that $(t+1)$ is also a lower bound on the number of rounds for interactive consistency. Moreover, as both consensus and interactive consistency algorithms presented in this chapter do not direct the processes to execute more than $(t+1)$ rounds, it follows that they are optimal with respect to the number of rounds.

This lower bound was first proved by M. Fischer and N. Lynch (1982). The following section presents a proof of it, which is due to M. Aguilera and S. Toueg (1999). The notion of valence used in the proof is due to M. Fischer, N.A. Lynch, and M.S. Paterson (1985).

10.3.1 Preliminary Assumptions and Definitions

Assumptions

- It is assumed that, in every round, each process broadcasts a message to all processes.

  It is easy to see this assumption does not limit the generality of the result. This is because, it is always possible to modify a round-based algorithm in order to obtain an equivalent algorithm using such a sending pattern. If during a round, a process sends a message m to a subset of the processes only, that message can carry the set of its destination processes and, when a process p_j receives m, it discards it if is is not a destination process.
- The lower bound proof considers the following assumptions. It is easy to see that, like the previous one, none of them limits the generality of the result.
  - The proof considers binary consensus.
  - The proof assumes that at least two processes do not crash (i.e., $t < n - 1$).
  - The proof assumes that there is one crash per round.
  - The proof considers the non-uniform version of consensus that is weaker than consensus (it requires only that no two correct processes decide different values).

Global state, valence, and k-round execution

- Considering an execution of a synchronous round-based algorithm A (a run), the *global state at the end round* r is made up of the state of each process at the end of this round (if a process crashed, its local state indicates the round at which it crashed).

  Let us notice that the global state at the end of a round is the same as the global state at the beginning of the next round. Only these global states need to be considered in the proof that follows. (A global state is sometimes called a *configuration*.)

  Given an initial global state and a failure pattern, the execution of an algorithm A gives rise to a sequence of global states.
- Let S be a global state obtained during the execution of a binary consensus algorithm A.
  - S is 0-*valent* (resp., 1-*valent*), if whatever the global states produced by A after S, the value 0 (resp., 1) only can be decided.
  - S is *univalent* if it is 0-valent or 1-valent.
  - S is *bivalent* if it not univalent.
- A k-*round execution* E_k of an algorithm A is an execution of A up to the end of round k.

  Let S_k be the corresponding global state. E_k is 0-valent, 1-valent, univalent or bivalent if S_k is 0-valent, 1-valent, univalent or bivalent, respectively.

10.3.2 The $(t + 1)$ Lower Bound

Theorem 42. *Let $t < n - 1$. Let us assume that at most one process crashes in each round. There is no round-based algorithm that solves binary consensus in t rounds in the system model $CSMP_{n,t}[\emptyset]$.*

Proof The proof is by contradiction. It supposes that there is an algorithm A that solves binary consensus in t rounds, in the presence of t process crashes (one per round). The proof follows from the two following lemmas that are proved in the next section.

- Lemma 39 shows that any $(t - 1)$-round execution E_{t-1} of A is univalent.
- Lemma 41 shows that A has a $(t - 1)$-round execution E_{t-1} that is bivalent.

These two lemmas contradict each other, thereby proving the impossibility for A to terminate in t rounds. Hence the $(t + 1)$ lower bound. $\square_{Theorem\ 42}$

10.3.3 Proof of the Lemmas

Lemma 39. *Any $(t-1)$-round execution E_{t-1} of A is univalent.*

Proof The proof is by contradiction. Let us assume that A has a bivalent $(t-1)$-round execution E_{t-1}. Let us consider the following three one-round extensions of E_{t-1} (Fig. 10.5).

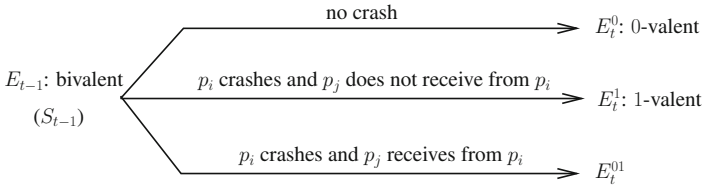

Figure 10.5: Three possible one-round extensions from E_{t-1}

- Let E_t^0 be the t-round execution obtained by extending E_{t-1} by one round in which no process crashes. As (by assumption) A terminates in t rounds, the correct processes decide by the end of round t of E_t^0. Let us suppose that they decide the value 0.
- As E_{t-1} is bivalent (contradiction assumption), it follows that it has a one-round extension E_t^1 in which the correct processes decide 1.

  Let us observe that in round t of E_t^1 exactly one process (say p_i) crashes. (At least one process crashes because otherwise E_t^0 and E_t^1 would be identical, and at most one process crashes because there is at most one crash per round.)

  Moreover, p_i must crash before sending its round t message to at least one correct process p_j, otherwise p_j would be unable to distinguish E_t^0 from E_t^1 and would consequently decide the same value in both executions.
- Let us now consider the one-round extension E_t^{01} that is identical to E_t^1 except that p_i sends its round t message to p_j. (This means the only difference between E_t^{01} and E_t^1 lies in the round t message from p_i to p_j that p_j receives in E_t^{01} and does not in E_t^1.)

Let p_k be a correct process different from p_j (such a process exists because $t < n-1$). We then have the following:

1. The correct process p_j cannot distinguish between E_t^0 and E_t^{01}. This is because, from its local state in S_{t-1} (its local state at the end of execution E_{t-1}), process p_j has received the same messages during the last round in both E_t^0 and E_t^{01}. Hence, it has to decide the same value in both executions. As it decides 0 in E_t^0, it has to decide 0 in E_t^{01}.
2. The correct process p_k cannot distinguish between E_t^1 and E_t^{01}. This is because (as previously for p_j) from its local state in S_{t-1}, it has received the same messages during the last round in both E_t^1 and E_t^{01}. Hence, it has to decide the same value in both executions. As it decides 1 in E_t^1, it has to decide 1 in E_t^{01}.

It follows that, while both p_j and p_k are correct in E_t^{01}, they decide differently, which contradicts the consensus agreement property and concludes the proof of the lemma. □*Lemma 39*

Lemma 40. *The algorithm A has a bivalent initial global state (or equivalently a bivalent 0-round execution).*

Proof The proof is by contradiction. Assuming that there is no bivalent initial global state, let $\mathcal{S}_0$ be the set of all 0-valent initial global states and $\mathcal{S}_1$ be the set of all 1-valent initial global states. As only 0 (resp., 1) can be decided when all processes propose 0 (resp., 1) the set $\mathcal{S}_0$ (resp., $\mathcal{S}_1$) is not empty. As these sets are not empty there must be two global states $S[0] \in \mathcal{S}_0$ and $S[1] \in \mathcal{S}_1$ that differ only in the value proposed by one process (say p_i).

Let us consider an execution E of A from $S[0]$ in which p_i crashes before taking any step. As $S[0]$ is 0-valent, it follows that the processes decide 0. But, as p_i does not participate in E, exactly the same execution can be produced from $S[1]$, and in this case the processes have to decide 1. In the execution E, no process can determine whether if the initial global state is $S[0]$ or $S[1]$. Consequently they have to decide the same value if E is executed from $S[0]$ or $S[1]$, contradicting the fact that $S[0]$ is 0-valent while $S[1]$ is 1-valent. □*Lemma 40*

Lemma 41. *The algorithm A has a bivalent $(t-1)$-round execution.*

Proof The proof shows that for each k, $0 \leq k \leq t-1$, there is a bivalent k-round execution E_k. It is based on an induction on k. The base case $k = 0$ is exactly what is proved by Lemma 40, namely, there is a bivalent initial global state S_0. The corresponding 0-round execution (in which no process has yet executed a step) is denoted E_0. So, let us consider the following induction assumption: for each k, $0 \leq k < t-1$, there is a bivalent k-round execution E_k.

To show that E_k can be extended by one round into a bivalent $(k+1)$-round execution E_{k+1}, the reasoning is by contradiction. Let us assume that every one-round extension of E_k is univalent. Let E^1_{k+1} be the one-round extension of E_k in which no process crashes during this round. Without loss of generality, let us assume that E^1_{k+1} is 1-valent. As E_k is bivalent, and all its one-round extensions are univalent, it has a one-round extension E^0_{k+1} that is 0-valent (Fig. 10.6).

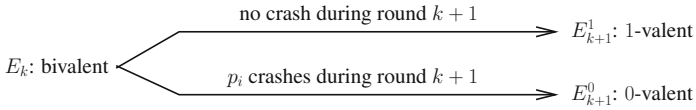

Figure 10.6: Extending the k-round execution E_k

As (a) E^1_{k+1} and E^0_{k+1} are one-round extensions of the same k-round execution E_k, (b) they have the different valence, and (c) no process crashes during the round $k+1$ of E^1_{k+1}, it follows that E^0_{k+1} is such that there is exactly one process (say p_i) that crashes during round $k+1$ ("exactly one" is because at most one process crashes per round), and fails to send its round $k+1$ message to some processes, say the processes $q_1, \ldots, q_m$ with $0 \leq m \leq n$ ($m = 0$ corresponds to the case where p_i crashes before it sent its round $k+1$ message to any process).

Starting from E^0_{k+1}, let us define a sequence of one-round extensions of E_k such that (see Table 10.2):

- $E_{k+1}[0]$ is E^0_{k+1} (hence, $E_{k+1}[0]$ is 0-valent), and
- $\forall\, j$, $0 < j \leq m$, $E_{k+1}[j]$ is identical to $E_{k+1}[j-1]$ except that p_i crashes after it has sent its round $k+1$ message to q_j. It is follows from this definition that p_i has sent its round $k+1$ message to the processes $q_1, \ldots, q_j$.

As by assumption all one-round extensions of E_k are univalent, $E_{k+1}[0]$, etc., until $E_{k+1}[m]$ are univalent.

Claim C. $\forall\, j$, $0 \leq j \leq m$, $E_{k+1}[j]$ is 0-valent.
Proof of the claim. The proof is by induction. As $E_{k+1}[0]$ is 0-valent, the claim follows for $j = 0$.

$(k+1)$-round execution	round $k+1$ message from p_i not sent to the processes
$E_{k+1}[0] \stackrel{\text{def}}{=} E^0_{k+1}$	$q_1, q_2, \ldots, q_j, q_{j+1}, \ldots, q_m$
$E_{k+1}[1]$	$q_2, \ldots, q_j, q_{j+1}, \ldots, q_m$
$E_{k+1}[j-1]$	$q_j, q_{j+1}, \ldots, q_m$
$E_{k+1}[j]$	$q_{j+1}, \ldots, q_m$
$E_{k+1}[m-1]$	q_m
$E_{k+1}[m]$	$\emptyset$

Table 10.2: Missing messages due to the crash of p_i

Hence, let us assume that all $(k+1)$-round executions $E_{k+1}[\ell]$, $0 \leq \ell < j$ are 0-valent, while $E_{k+1}[j]$ is 1-valent. We show that it is not possible.

Let us extend (see Fig. 10.7) the 0-valent execution $E_{k+1}[j-1]$ into the execution E^0_{k+2} and the 1-valent execution $E_{k+1}[j]$ into the execution E^1_{k+2} by crashing, in both executions, process q_j at the very beginning of round $k+2$ (if it has not crashed before). It follows there is no round after round $k+1$ in which q_j sends a message. Let us notice that, as $k < t-1$, round $k+2$ exists.

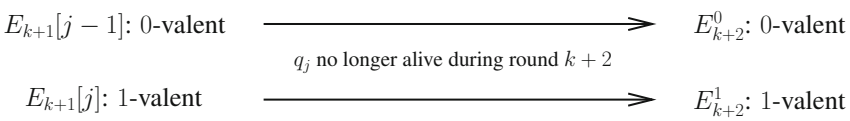

Figure 10.7: Extending two $(k+1)$-round executions

Let us observe that no process that has not crashed by the end of round $k+2$ can distinguish E^0_{k+2} from E^1_{k+2} (any such process has the same local state in both executions). Hence, E^0_{k+2} and E^1_{k+2} are identical for the processes that terminate round $k+2$. Hence, these processes have to decide both 0 (because E^0_{k+2} is 0-valent), and 1 (because E^1_{k+2} is 1-valent), which is clearly impossible. End of proof of the claim.

It follows from the claim that $E_{k+1}[m]$ is 0-valent. Let us now consider E^1_{k+1} that is 1-valent. The only difference between these two $(k+1)$-round executions is that p_i crashes at the end of the round $(k+1)$ in $E_{k+1}[m]$, and does not crash during the round $(k+1)$ in E^1_{k+1}. Let us construct the two following $(k+2)$-round executions (Fig. 10.8).

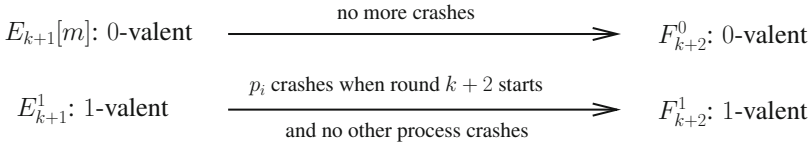

Figure 10.8: Extending again two $(k+1)$-round executions

- Let F^1_{k+2} be the one-round extension of E^1_{k+1} where p_i crashes when round $k+2$ starts and then no other process crashes. Let us notice that F^1_{k+2} is 1-valent.
- Let F^0_{k+2} be the one-round extension of $E_{k+1}[m]$, where no more process crashes. Let us notice that F^0_{k+2} is 0-valent.

Let us observe on the one hand that a correct process has to decide 1 from the $(k+2)$-round execution F^1_{k+2}, and 0 from the $(k+2)$-round execution F^0_{k+2}. On the other hand, no process executing round

$k+2$ can distinguish if the execution is F^1_{k+2} or F^0_{k+2}; hence, it has to decide both 0 and 1 which is impossible. A contradiction which concludes the proof of the lemma. $\square_{Lemma\ 41}$

10.4 Summary

This chapter introduced two basic agreement abstractions, namely consensus and interactive consistency (also called vector consensus). In each of them, each process proposes a value. While consensus allows processes to agree on one of the values they propose, interactive consistency allows them to agree on a vector, with one entry per process, such that entry i contains the value v_i proposed by p_i if this process is correct, and v_i or $\perp$ if it is faulty. These definitions are suited to both synchronous and asynchronous systems.

The chapter then presented round-based algorithms, that implement these agreement abstractions in the system model $CSMP_{n,t}[\emptyset]$, i.e., synchronous message-passing systems in which any number $t < n$ of processes may crash. It was also shown that $(t+1)$ is a lower bound on the number of rounds to implement these agreement abstractions in $CSMP_{n,t}[\emptyset]$.

10.5 Bibliographic Notes

- The message-passing synchronous model with process crash failures was introduced in Chap. 1. It is also presented in textbooks such as [43, 185, 271, 367]). Lots of synchronous algorithms for failure-free systems are presented in [368].
- The consensus agreement abstraction originated in the work of L. Lamport, R. Shostask, and M. Pease [258, 263, 342], who also defined the Byzantine failure model, the Byzantine generals problem, and the interactive consistency agreement abstraction. These papers established lower bounds on the number of rounds to solve this problem in the context of synchronous systems prone to Byzantine failures and presented corresponding algorithms.
- All the algorithms presented in this chapter are based on variants of the extinction/propagation strategy, namely, during every round, each process propagates the new values it learned during the previous round. Similar distributed algorithms are described in many textbooks such as [43, 185, 250, 271, 362, 366, 367, 368].
- The notion of an atomic process failure is due C. Delporte, H. Fauconnier, R. Guerraoui and B. Pochon [124].
- The $(t+1)$ lower bound for consensus and interactive consistency was first been proved for the Byzantine failure model in the early eighties [136, 161, 262]. Proofs customized for the process crash failure model appeared later (e.g., in [21, 135, 143, 271, 299]). The proof presented in this chapter is due Aguilera and Toueg [21].
- The notion of valence is due to M. Fischer, N. A. Lynch, and M. S. Paterson [162]. This notion was introduced to prove the impossibility of consensus in the asynchronous model $CAMP_{n,t}[\emptyset]$.

10.6 Exercises and Problems

1. Let us assume an algorithm A that implements interactive consistency in the asynchronous system model $CAMP_{n,t}[\emptyset]$. Design an algorithm that builds a perfect failure detector in the system model $CAMP_{n,t}[A]$ ($CAMP_{n,t}[\emptyset]$ enriched with A).

   Solution in [211].

2. Let $CSMP_{n,t}[\text{SO}]$ be the system model $CSMP_{n,t}[\emptyset]$ weakened as follows: a faulty process is a process that crashes, or a process that forgets to send messages. Hence, a faulty process can

never crash, but the message it is assumed to broadcast during a round can be received by an arbitrary subset of process. This failure model is called the *send omission* failure model.

Design and proof a consensus algorithm suited to the model $CSMP_{n,t}[\text{SO}]$.

Solution in Chapter 7 of [367].

3. Let $CSMP_{n,t}[\text{GO}]$ be the system model $CSMP_{n,t}[\emptyset]$ weakened as follows: a faulty process is a process that crashes, or a process that forgets to send or receive messages. This is the *general omission* failure model.
   - Show that the model constraint $t < n/2$ is a necessary condition to solve consensus in the system model $CSMP_{n,t}[\text{GO}]$. (Hint: partition the set of processes in two subsets Q_1 and Q_2 of size $\lceil \frac{n}{2} \rceil$, and $\lfloor \frac{n}{2} \rfloor$, and consider the case where, while no process crashes, all the processes of Q_2 commit send and receive omission failures with respect to the processes of Q_1.)

     Remark. The proof is based on an indistinguishability argument as already used in the proofs of some theorems (e.g., Theorem 9 and Theorem 18).

     Solution in Chapter 7 of [367].
   - Design and proof a consensus algorithm for the system model $CSMP_{n,t}[\text{GO}]$. As a faulty process may not crash, and may remain isolated from the correct processes, it cannot decide the value decided by the correct processes. In this case, it is allowed to decide a special default value denoted $\perp$. Hence, if a process, that does not crash, decides $\perp$, it knows that it is faulty.

     Let us remark that the existence of such an algorithm, shows that the model constraint $t < n/2$ is sufficient to solve consensus in $CSMP_{n,t}[\text{GO}]$.

     Solution in Chapter 7 of [356].