Revisiting Immediate Snapshot

Carole Delporte\textsuperscript{1}, Hugues Fauconnier\textsuperscript{1}, Sergio Rajsbaum\textsuperscript{2}, Michel Raynal\textsuperscript{3}

\textsuperscript{1} IRIF, Université Paris Diderot, Paris, France
\textsuperscript{2} Instituto de Matemáticas, UNAM, México D.F. 04510, México
\textsuperscript{3} IUF & IRISA (Université de Rennes), Rennes, France

Abstract. An immediate snapshot object is a high level communication object, built on top of a read/write distributed system in which all except one processes may crash. It allows each process to write a value and obtains a set of pairs (process id, value) such that, despite process crashes and asynchrony, the sets obtained by the processes satisfy noteworthy inclusion properties. Considering an \(n\)-process model in which up to \(t\) processes are allowed to crash (\(t\)-crash system model), this paper is on the construction of \(t\)-resilient immediate snapshot objects. In the \(t\)-crash system model, a process can obtain values from at least \((n - t)\) processes, and, consequently, \(t\)-immediate snapshot is assumed to have the properties of the basic \((n - 1)\)-resilient immediate snapshot plus the additional property stating that each process obtains values from at least \((n - t)\) processes. The main result of the paper is the following. While there is a (deterministic) \((n - 1)\)-resilient algorithm implementing the basic \((n - 1)\)-immediate snapshot in an \((n - 1)\)-crash read/write system, there is no \(t\)-resilient algorithm in a \(t\)-crash read/write model when \(t \in [1..(n - 2)]\). This means that, when \(t < n - 1\), the notion of \(t\)-resilience is inoperative when one has to implement \(t\)-immediate snapshot for these values of \(t\): the model assumption “at most \(t\) processes may crash” does not provide us with additional computational power allowing for the design of a genuine \(t\)-resilient algorithm (genuine meaning that such an algorithm would work in the \(t\)-crash model, but not in the \((t + 1)\)-crash model). To show these results, the paper relies on well-known distributed computing agreement problems such as consensus and \(k\)-set agreement.

Keywords: Asynchronous system, Atomic read/write register, Consensus, Distributed computability, Immediate snapshot, Impossibility, Iterated model, \(k\)-Set Agreement, Linearizability, Process crash failure, Snapshot object, \(t\)-Resilience, Wait-freedom.

1 Introduction

Immediate snapshot object and iterated immediate snapshot model

The immediate snapshot (IS) communication object and the associated iterated immediate snapshot (IIS) model have been introduced in [5,20], and later investigated in [7]. This distributed computing model consists of \(n\) asynchronous processes, among which any subset of up to \((n - 1)\) processes may crash\textsuperscript{1}, which execute a sequence of asyn-

\textsuperscript{1} From a terminology point of view, we say \(t\)-failure model (in the present case \(t\)-crash model) if the model allows up to \(t\) processes to fail. We keep the term \(t\)-resilience for algorithms.
chronous rounds. One and only one immediate snapshot (IS) object is associated with each round, which allows the processes to communicate during this round. More precisely, for any $x > 0$, a process accesses the $x$-th immediate snapshot only when it executes the $x$-th round, and it accesses it only once.

From an abstract point of view, an IS object $IMSP$, can be seen as an initially empty set, which can then contain at most $n$ pairs (one per process), each made up of a process index and a value. This object provides the processes with a single operation denoted $write_{\text{snapshot}}()$, that each process may invoke only once. The invocation $IMSP.write_{\text{snapshot}}(v)$ by a process $p_i$ adds the pair $(i, v)$ to $IMSP$ and returns a set of pairs belonging to $IMSP$ such that the sets returned to the processes that invoke $write_{\text{snapshot}}()$ satisfy specific inclusion properties. It is important to notice that, in the IIS model, the processes access the sequence of IS objects one after the other, in the same order, and asynchronously.

The noteworthy feature of the IIS model is the following. It has been shown by Borowsky and Gafni in [7], that this model is equivalent to the usual read/write wait-free model ($(n-1)$-crash model) for task solvability with the wait-freedom progress condition (any non-faulty process obtains a result). Its advantage lies in the fact that its runs are more structured and easier to analyze than the runs in the basic read/write shared memory model [27]. It is also the basis of the combinatorial topology approach for distributed computing (e.g., [17]). Hence, IS objects constitute the algorithmic foundation of distributed iterated computing models.

It has been shown in [30] that trying to enrich the IIS model with (non trivial) failure detectors is inoperative. This means that, for example, enriching IIS with the failure detector $\Omega$ (which is the weakest failure detector that allows consensus to be solved in the basic read/write communication model [10,24]) does not allow to solve consensus in such an enriched IIS model. However, it has been shown in [29] that it is possible to capture the power of a failure detector (and other partially synchronous systems) in the IIS model by appropriately restricting its set of runs, giving rise to the Iterated Restricted Immediate Snapshot (IRIS) model. This approach has been further investigated in [32].

The IIS model has many interesting features among which the following two are noteworthy. The first is on the foundation side of distributed computing, namely IIS established a strong connection linking distributed computing and algebraic topology (see [6,17,19,21,33]). The second one lies on the algorithmic and programming side, namely IIS allows for a recursive formulation of algorithms solving distributed computing problems. This direction, initiated in [5,15], has also been investigated in [28,31].

Another line of research is investigated in [14]. This paper considers models of distributed computations defined as subsets of the runs of the iterated immediate snapshot model. In such a context, it uses topological techniques to identify the tasks that are solvable in such a model.

The $(n-1)$-crash model is also called wait-free model [16]. Several progress conditions have been associated with $(n-1)$-resilient algorithms: wait-freedom [16], non-blocking [22], or obstruction-freedom [18]. (See a unified presentation in Chapter 5 of [31].)
t-Crash model and t-resilient algorithms

The previous basic read/write model and IIS model consider that all but one process may crash. Differently, a t-crash model assumes that at most t processes may crash, i.e., by assumption, at least \((n - t)\) of them never crash. As already said, an algorithm designed for such a model is said to be t-resilient.

One of the most fundamental results of distributed computing is the impossibility to design a 1-resilient consensus algorithm in the 1-crash \(n\)-process model, be the communication medium an asynchronous message-passing system [13] or a read/write shared memory [25]. Differently, other problems, such as renaming (introduced in the context of t-resilient message-passing systems where \(t < n/2\) [3]), can be solved by \((n - 1)\)-resilient algorithms in the \((n - 1)\)crash read/write shared memory model (such renaming algorithms are described in several textbooks, e.g. [4,31,34]).

Contribution of the paper

When considering the t-crash \(n\)-process model where \(t < n - 1\), and assuming that each correct process writes a value, a process may wait for values written by \((n - t)\) processes without risking being blocked forever. This naturally leads to the notion of a t-crash \(n\)-process iterated model, generalizing the IIS model to any value of \(t\). To this end the paper introduces the notion of a \(k\)-immediate snapshot object, which generalizes the basic \((n - 1)\)-immediate snapshot object. More precisely, when considering a \(t\)-immediate snapshot object in a \(t\)-crash \(n\)-process model, an invocation of write\_snapshot() by a process returns a set including at least \((n - t)\) pairs (while it would return a set of \(x\) pairs with \(1 \leq x \leq n\) if the object was an IS object). Hence, a \(t\)-immediate snapshot object allows processes to obtain as much information as possible from the other processes while guaranteeing progress.

The obvious question is then the implementability of a \(t\)-immediate snapshot object in the \(t\)-crash \(n\)-process model. This question is answered in this paper, which shows that it is impossible to implement a \(t\)-IS object in a \(t\)-crash \(n\)-process model when \(0 < t < n - 1\). More precisely we prove that implementing a \(t\)-IS object is equivalent\(^2\) to implementing consensus when \(t < n/2\) and enables to implement \((2t - n + 2)\)-set agreement when \(n/2 \leq t < n - 1\).

At first glance, this impossibility result may seem surprising. An IS object is a snapshot object (a) whose operations write() and snapshot() are glued together in a single operation write\_snapshot(), and (b) satisfying an additional property linking the sets of pairs returned by concurrent invocations (called Immediacy property, Section 2.2). Then, as already indicated, a \(t\)-IS object is an IS object such that the sets returned by write\_snapshot() contain at least \((n - t)\) pairs (Output size property, Section 2.4). The same Output size property on the sets returned by a snapshot object can be trivially implemented in a \(t\)-crash \(n\)-process model. Let us call \(t\)-snapshot such a constrained snapshot object. Hence, while a \(t\)-snapshot object can be implemented in the \(t\)-crash \(n\)-process model, a \(t\)-IS object cannot when \(0 < t < n - 1\).

Roadmap

As previously indicated, the paper is on the computability power of \(t\)-IS objects in the

\[^2\] A is equivalent to B if A can be (computationally) reduced to B and reciprocally.
t-crash computing model, for $t < n - 1$. Made up of 7 sections, it has the following content.

- Section 2 introduces the basic crash-prone read/write system model, immediate snapshot, a $k$-set agreement, and $k$-immediate snapshot ($k$-IS). It also proves a theorem which captures the additional computational power of $k$-immediate snapshot with respect to the basic $(n - 1)$-immediate snapshot.
- Assuming a majority of processes never crash, i.e. a $t$-crash read/write model in which $t < n/2$, Section 3 shows that it is impossible to implement $t$-immediate snapshot in such a model. The proof is a reduction of the consensus problem to $t$-immediate snapshot.
- Assuming $t \leq n - 1$, Section 4 presents a reduction of $t$-immediate snapshot to consensus in a $t$-crash read/write model. When combined with the result of Section 3, this shows that $t$-immediate snapshot and consensus have the same computational power in any $t$-crash model where $t < n/2$.
- Assuming a $t$-crash read/write model in which $n/2 \leq t < n - 1$, Section 5 shows that it is impossible to implement $t$-immediate snapshot in such a model. The proof is a reduction of the $(2t - n + 2)$-set agreement problem to $t$-immediate snapshot.
- By a simulation argument, Section 6 shows that consensus is not solvable with $t$-immediate snapshot when $n/2 \leq t < n$ proving that the computational power of $t$-immediate snapshot when $0 < t < n/2$ is strictly stronger than the computational power of $t$-immediate snapshot when $n/2 \leq t < n$.

Finally, Section 7 concludes the paper.

2 Immediate Snapshot, $k$-Set Agreement, and $k$-Immediate Snapshot

2.1 Basic read/write system model

Processes

The computing model is composed of a set of $n \geq 3$ sequential processes denoted $p_1$, ..., $p_n$. Each process is asynchronous which means that it proceeds at its own speed, which can be arbitrary and remains always unknown to the other processes.

A process may halt prematurely (crash failure), but executes correctly its local algorithm until it possibly crashes. The model parameter $t$ denotes the maximal number of processes that may crash in a run. A process that crashes in a run is said to be faulty. Otherwise, it is correct or non-faulty. Let us notice that, as a faulty process behaves correctly until it crashes, no process knows if it is correct or faulty. Moreover, due to process asynchrony, no process can know if another process crashed or is only very slow.

It is assumed that (a) $0 < t < n$ (at least one process may crash and at least one process does not crash), and (b) any process, until it possibly crashes, executes the algorithm assigned to it.
Communication layer

The processes cooperate by reading and writing Single-Write Multi-Reader (SWMR) atomic read/write registers [23]. This means that the shared memory can be seen as a set of arrays \( A[1..n] \) where, while \( A[i] \) can be read by all processes, it can be written only by \( p_i \).

Notation

The previous model is denoted \( CARW_{n,t}[\emptyset] \) (which stands for “Crash Asynchronous Read/Write with \( n \) processes, among which up to \( t \) may crash”). A model constrained by a predicate on \( t \) (e.g. \( t < x \)) is denoted \( CARW_{n,t}[t < x] \). Hence, as we assume at least one process does not crash, \( CARW_{n,t}[t < n] \) is a synonym of \( CARW_{n,t}[\emptyset] \), which (as always indicated) is called wait-free model. When considering \( t \)-crash models, \( CARW_{n,t}[t \leq \alpha] \) is less constrained than \( CARW_{n,t}[t < \alpha - 1] \).

Shared objects are denoted with capital letters. The local variables of a process \( p_i \) are denoted with lower case letters, sometimes suffixed by the process index \( i \).

2.2 One-shot immediate snapshot object

The immediate snapshot (IS) object was informally presented in the introduction. It can be seen as a variant of the snapshot object introduced in [1,2]. While a snapshot object provides the processes with two operations (\( \text{write}() \) and \( \text{snapshot}() \)) which can be invoked separately by a process (usually \( \text{write}() \) before \( \text{snapshot}() \)), an immediate snapshot provides the processes with a single operation \( \text{write\_snapshot}() \). One-shot means that a process may invoke \( \text{write\_snapshot}() \) at most once.

Definition

An IS object \( IMSP \) is a set, initially empty, that will contain pairs made up of a process index and a value. Let us consider a process \( p_i \) that invokes \( IMSP.\text{write\_snapshot}(v) \). This invocation adds the pair \( \langle i, v \rangle \) to \( IMSP \) (contribution of \( p_i \) to \( IMSP \)), and returns to \( p_i \) a set, called view and denoted \( \text{view}_i \), such that the sets returned to the processes collectively satisfy the following properties.

- Termination. The invocation of \( \text{write\_snapshot}() \) by a correct process terminates.
- Self-inclusion. \( \forall i: \langle i, v \rangle \in \text{view}_i \).
- Validity. \( \forall i: (\langle j, v \rangle \in \text{view}_i) \Rightarrow p_j \text{ invoked } \text{write\_snapshot}(v) \).
- Containment. \( \forall i, j: (\text{view}_i \subseteq \text{view}_j) \lor (\text{view}_j \subseteq \text{view}_i) \).
- Immediacy. \( \forall i, j: (\langle i, v \rangle \in \text{view}_j) \Rightarrow (\text{view}_i \subseteq \text{view}_j) \).

It is relatively easy to show that the Immediacy property can be re-stated as follows: \( \forall i, j: ((i, -) \in \text{view}_j) \land ((j, -) \in \text{view}_i) \Rightarrow (\text{view}_i = \text{view}_j) \).

Implementation

Implementations of an IS object in the wait-free model \( CARW_{n,t}[0 < t < n] \) are described in [5,15,28,31]. While both a one-shot snapshot object and an IS object satisfy the Self-inclusion, Validity and Containment properties, only an IS object satisfies the Immediacy property. This additional property creates an important difference, from
which follows that, while a snapshot object is atomic (operations on a snapshot object can be linearized [22]), an IS object is not atomic (its operations cannot always be linearized). However, an IS object is set-linearizable (set-linearizability allows several operations to be linearized at the same point of the time line [9,26]).

The iterated immediate snapshot (IIS) model
In this model (introduced in [7]), the shared memory is composed of a (possibly infinite) sequence of IS objects: IMSP[1], IMSP[2], ... These objects are accessed sequentially and asynchronously by the processes according to the following round-based pattern executed by each process \( p_i \). The variable \( r_i \) is local to \( p_i \); it denotes its current round number.

\[
\begin{align*}
  r_i & \leftarrow 0; \ell s_i \leftarrow \text{initial local state of } p_i \text{ (including its input, if any)}; \\
  \text{repeat forever} & \% \text{asynchronous IS-based rounds} \\
  r_i & \leftarrow r_i + 1; \\
  \text{view}_i & \leftarrow \text{IMSP}[r_i].\text{write\_snapshot}(\ell s_i); \\
  \text{computation of a new local state } \ell s_i \text{ (which contains } \text{view}_i) \\
  \text{end repeat.}
\end{align*}
\]

As indicated in the Introduction, when considering distributed tasks (as formally defined in [8,21]), the IIS model and \( \mathcal{CARW}_{n,t}[0 < t < n] \) have the same computational power [7].

2.3 \( k \)-Set agreement

\( k \)-Set agreement was introduced by S. Chaudhuri [11] to investigate the relation linking the number of different values that can be decided in an agreement problem, and the maximal number of faulty processes. It generalizes consensus which corresponds to the case \( k = 1 \).

A \( k \)-set agreement object is a one-shot object that provides the processes with a single operation denoted \( \text{propose}_k() \). This operation allows the invoking process \( p_i \) to propose a value it passes as an input parameter (called \( \text{proposed} \) value), and obtain a value (called \( \text{decided} \) value). The object is defined by the following set of properties.

- Termination. The invocation of \( \text{propose}_k() \) by a correct process terminates.
- Validity. A decided value is a proposed value.
- Agreement. No more than \( k \) different values are decided.

It is shown in [6,21,33] that the problem is impossible to solve in \( \mathcal{CARW}_{n,t}[k \leq t] \).

2.4 \( k \)-Immediate Snapshot

A \( k \)-immediate snapshot object (denoted \( k \)-IS) is an immediate snapshot object with the following additional property.

- Output size. The set \( \text{view} \) obtained by a process is such that \( |\text{view}| \geq n - k \).

Theorem 1. A \( k \)-IS object cannot be implemented in \( \mathcal{CARW}_{n,t}[k < t] \).
Proof To satisfy the output size property, the view obtained by a process \( p_i \) must contain pairs from \((n - k)\) different processes. If \( t \) processes crash (e.g., initially), a process can obtain at most \((n - t)\) pairs. If \( t > k \), we have \( n - t < n - k \). It follows that, after it has obtained pairs from \((n - t)\) processes, a process can remain blocked forever waiting for the \((t - k)\) missing pairs.

\[\text{Proof} \]

Theorem 1 Considering the system model \( CARW_{n,t}[0 < t < n - 1] \), the next theorem characterizes the power of a \( t\)-IS object in term of the Containment property.

Theorem 2. Considering the system model \( CARW_{n,t}[0 < t < n - 1] \), and a \( t \)-IS object, let us assume that all correct processes invoke \( \text{write\_snapshot}() \). No process obtains a view with less than \((n - t)\) pairs. Moreover, if the size of the smallest view obtained by a process is \( \ell (\ell \geq n - t) \), there is a set \( S \) of processes such that \(|S| = \ell \geq n - t\) and each process of \( S \) obtains the smallest view or crashes during its invocation of \( \text{write\_snapshot}() \).

\[\text{Proof} \]

It follows from the Output size property of the \( t \)-IS object that no view contains less than \((n - t)\) pairs. Let \( \text{view} \) be the smallest view returned by a process, and let \( \ell = |\text{view}| \). We have \( \ell \geq n - t \). Moreover, due to (a) the Immediacy property (namely \(((i, -) \in \text{view}) \Rightarrow \text{view}_{i} \subseteq \text{view} \)) and (b) the minimality of \( \text{view} \), it follows that \( \text{view}_{i} = \text{view} \). As this is true for each process whose pair participates in \( \text{view} \), and \( t = |\text{view}| \), it follows that there is a set \( S \) of processes such that \(|S| = \ell \geq n - t\) and each of its processes obtains the view \( \text{view} \), or crashed during its invocation of \( \text{write\_snapshot}() \). Due to the Containment property, the others processes crash or obtain views which strictly include \( \text{view} \).

\[\text{Proof} \]

Theorem 2

3 \( t \)-Immediate Snapshot is Impossible in \( CARW_{n,t}[0 < t < n/2] \)

This section shows that it is impossible to implement a \( t \)-IS object when \( 0 < t < n/2 \).

From \( t \)-IS to consensus in \( CARW_{n,t}[0 < t < n/2] \)

Algorithm 1 reduces consensus to \( t \)-IS in the system model \( CARW_{n,t}[0 < t < n/2] \). As at most \( t < n/2 \) process may crash, at least \( n - t > n/2 \) processes invoke the consensus operation \( \text{propose}(v) \).

Algorithm 1: Solving consensus in \( CARW_{n,t}[0 < t < n/2, t\text{-IS}] \) (code for \( p_i \)
In addition to a \( t\)-IS object denoted \( IMSP \), the processes access an array \( VIEW[1..n] \) of SWMR atomic registers, initialized to \([\bot, \ldots, \bot]\). The aim of \( VIEW[i] \) is to store the view obtained by \( p_i \) from the \( t\)-IS object \( IMSP \).

When it calls \( \text{propose}_i(v) \), a process \( p_i \) invokes first the \( t\)-IS object, in which it deposits the pair \((i, v)\), and obtains a view from it, that it writes in \( VIEW[i] \) to make it publicly known (line 1). Then, it waits (line 2) until it sees the views of at least \((t+1)\) processes (as \( n-t \geq t+1 \), \( p_i \) cannot block forever and at least one of these views is from a correct process). Process \( p_i \) extracts then of these views the one with the smallest cardinality (line 3), and finally returns proposed value contained in this smallest view (line 4).

**Theorem 3.** Algorithm 1 reduces consensus to \( t\)-IS in \( \text{CARW}_{n,t}[0 < t < n/2] \).

**Proof** Let us first prove the consensus Termination property. As \( n-t \geq t+1 \), and there are at least \((n-t)\) correct processes, it follows that at least \((n-t)\) entries of \( VIEW[1..n] \) are eventually different from \( \bot \). Hence, no correct process can remain blocked forever at line 2, which proves consensus Termination.

Let us now consider the consensus Agreement property. It follows from Theorem 2 that there is a set of at least \( \ell \geq n-t \) processes, that obtained the same view \( \min_{\text{view}} \) (or crashed before returning from \( \text{write\_snapshot}() \)), and this view is the smallest view obtained by a process and its size is \(|\min_{\text{view}}| = \ell \). As \( \ell \geq n-t \) and \((n-t)+(t+1) > n \), it follows from the waiting predicate of line 2, that, any process that executes line 3, obtains a copy of \( \min_{\text{view}} \), and consequently we have \( \text{view} = \min_{\text{view}} \) at line 3. It follows that no two processes can decide different values.

Finally, the consensus Validity property follows from the fact that any pair contained in a view is composed of a process index and the value proposed by the corresponding process.

**Corollary 1.** Implementing a \( t\)-IS object in \( \text{CARW}_{n,t}[0 < t < n/2] \) is impossible.

**Proof** The proof is an immediate consequence of Lemma 3, and the fact that consensus cannot be solved in \( \text{CARW}_{n,t}[0 < t < n/2] \) [25].

**4 From Consensus to \( t\)-IS in \( \text{CARW}_{n,t}[0 < t \leq n-1] \)**

Algorithm 2 describes a reduction of \( t\)-IS to consensus in \( \text{CARW}_{n,t}[0 < t \leq n-1] \). This algorithm uses two shared data structures. The first is an array \( REG[1..n] \) of SWMR atomic registers (where \( REG[i] \) is associated with \( p_i \)). The second is an array of \((t+1)\) consensus objects denoted \( CONS[(n-t)..n] \).

The invocation of \( \text{write\_snapshot}(v_i) \) by a process \( p_i \) deposits \( v_i \) in \( REG[i] \), and launches two underlying tasks \( T1 \) and \( T2 \). The task \( T2 \) is a simple waiting task, which will return a view to the calling process \( p_i \). The \( \text{return\() \) statement at line 9 terminates the \( \text{write\_snapshot()} \) operation invoked by \( p_i \). The termination of \( T2 \) does not kill the task \( T1 \) which may continue executing.
Algorithm 2: Implementing $t$-IS in CARW$_{n,t}[0 < t < n/2, \text{CONS}]$ (code for $p_i$)

Task $T1$ (lines 2-8) has two aims: provide $p_i$ with a view $\text{view}_i$ (line 6), and prevent processes from deadlocking, thereby allowing them to terminate. It consists in a loop that is executed $(t+1)$ times. The aim of the $k$-th iteration (starting at $k = 0$) is to allow processes to obtain a view including $(n - t + k)$ pairs. More precisely, we have the following.

- When it enters the $k$-th iteration, a process $p_i$ first waits until it obtains a set of pairs, denoted $\text{aux}_i$, which (a) contains $(n - t + k)$ pairs, (b) contains the set set of pairs $\text{dec}_i$ decided during the previous iteration, and (c) contains only pairs extracted from the array $\text{REG}[1..n]$. This is captured by the predicate of line 4.
- Then, $p_i$ proposes the set $\text{aux}_i$ to the consensus object $\text{CONS}[n - t + k]$ associated with the current iteration step (line 5). The set decided is stored in $\text{dec}_i$.
- Finally, if its pair $\langle i, v_i \rangle$ belongs to $\text{dec}_i$ and $p_i$ has not yet decided (i.e., no set has yet been assigned to $\text{view}_i$), it does it by writing $\text{dec}_i$ in $\text{view}_i$. Let us notice that this ensures the Self-inclusion property of the $t$-IS object. Moreover, a process decides no more than once.

Whether a process decides or not during the current iteration step, it systematically proceeds to the next iteration step. Hence, a process that obtains its view during an iteration step $x$ can help other processes to obtain a view during later iteration steps $y > x$.

**Theorem 4.** Algorithm 2 reduces $t$-IS to consensus in CARW$_{n,t}[0 < t < n/2, \text{CONS}]$.

**Proof** The Self-inclusion property follows directly from the predicate $\langle i, v_i \rangle \in \text{dec}_i$ used before assigning $\text{dec}_i$ to $\text{view}_i$ at line 6.

The Validity property follows from (a) the fact that a process $p_i$ assigns the value it wants to deposit in the $t$-IS object in $\text{REG}[i]$, (b) this atomic variable is written at most once (line 1), and (c) the predicate $\text{REG}[j] \neq \bot$ is used at line 4 to extract values from $\text{REG}[1..n]$. 

---

**Algorithm 2**

```plaintext
operation write_snapshot($v_i$) is
(1) $\text{REG}[i] \leftarrow v_i$; $\text{view}_i \leftarrow \emptyset$; $\text{dec}_i \leftarrow \emptyset$; $k \leftarrow -1$; launch the tasks $T1$ and $T2$.

(2) task $T1$ is
(3) repeat $k \leftarrow k + 1$;
(4) wait $(\exists \text{ a set } \text{aux}_i : (\text{dec}_i \subseteq \text{aux}_i) \land (|\text{aux}_i| = n - t + k) \land (\text{aux}_i \subseteq \{ \langle j, \text{REG}[j] \rangle \text{ such that } \text{REG}[j] \neq \bot \})$);
(5) $\text{dec}_i \leftarrow \text{CONS}[n - t + k].\text{propose}_i(\text{aux}_i)$;
(6) if $(\langle i, v_i \rangle \in \text{dec}_i) \land (\text{view}_i = \emptyset)$ then $\text{view}_i \leftarrow \text{dec}_i$ end if
(7) until $(k = t)$ end repeat
(8) end task $T1$.

(9) task $T2$ is $\text{wait} (\text{view}_i \neq \emptyset)$; return ($\text{view}_i$) end task $T2$.
end operation.
```
The Output size property follows from the predicate of line 4, which requires that any set $aux_i$ (and consequently any set $dec_i$ output by a consensus object) contains at least $(n-t)$ pairs.

To prove the Immediacy property, let us consider any two processes $p_i$ and $p_j$ such that $\langle j, v_j \rangle \in view_i$ and $\langle i, v_i \rangle \in view_j$. Let $dec_x[k]$ denote the local variable $dec_x$ after $p_x$ assigned it a value at line 5 during iteration step $k$.

Let $k_i$ be the iteration step at which $p_i$ assigns $dec_i$ to $view_i$ (due to the predicate $view_i = \emptyset$ used at line 5, such an assignment is done only once). It follows from the first predicate of line 6, that $\langle i, v_i \rangle \in dec_i[k_i] = view_i$ (otherwise, $view_i$ would not be assigned $dec_i$ during iteration step $k$). Hence, $k_i$ is the only iteration step at which we have $\langle i, v_i \rangle \in dec_x[k] = view_x$. Let $k_j$ be the iteration step at which $p_j$ assigns $dec_j$ to $view_j$. Hence, $k_j$ is the only iteration step at which we have $\langle j, v_j \rangle \in dec_j[k_j] = view_j$.

As far as the Termination property is concerned, let $p$ be the number of processes that have deposited a value in $REG[1..n]$. We have $n-t \leq p \leq n$. It follows from the predicate in the wait statement (line 4), that no process can block forever at this line for $k \in [0..p-n+t]$. As there are at least $(n-t)$ correct processes, and none of them can be blocked forever at line 4, it follows that each of them invokes $CONS[n-t+k].propose_i()$ (line 5), for each $k \in [0..p-n+t]$. Hence, the only reason for a correct process not to obtain a view (and terminate), is to never execute the assignment $view_i \leftarrow dec_i$ at line 7.

The sequence of consensus instances outputs a sequence of sets of pairs whose successive sizes are $(n-t), (n-t+1), \ldots, p$, which means that the identity of every of the $p$ processes that wrote in $REG[1..n]$ appears at least once in the sequence of consensus outputs. Hence, for each correct process $p_i$, there is a consensus instance...
whose output \( \text{dec} \) is such that, while \( \text{view}_i = \emptyset \), we have \( \langle i, v_i \rangle \in \text{dec} \), which concludes the proof of the Termination property.

\[ \square \text{Theorem 4} \]

**Corollary 2.** Consensus and \( t \)-IS are equivalent in \( \mathcal{CARW}_{n,t}[0 < t < n/2] \).

**Proof**  The proof follows from Theorem 3 (Algorithm 1) and Theorem 4 (Algorithm 2).

\[ \square \text{Theorem 2} \]

### 5 \( t \)-Immediate Snapshot is Impossible in \( \mathcal{CARW}_{n,t}[n/2 \leq t < n - 1] \)

This section shows that it is impossible to implement a \( t \)-IS object in \( \mathcal{CARW}_{n,t}[n/2 \leq t < n - 1] \). To this end, it presents a reduction of \( k \)-set agreement (in short \( k \)-SA) to \( t \)-IS for \( k = 2t - n + 2 \) (e.g., a reduction of \((n - 2)\)-SA agreement to \((n - 2)\)-IS in \( \mathcal{CARW}_{n,t}[t = n - 2] \)).

**From \( t \)-IS to \( (2t - k + 2)\)-set agreement in \( \mathcal{CARW}_{n,t}[n/2 \leq t < n - 1, t \text{-IS}] \)**

Algorithm 3 reduces \((2t - n + 2)\)-set agreement to \( t \)-IS in \( \mathcal{CARW}_{n,t}[n/2 \leq t < n - 1] \).

As at most \( t \) process may crash, at least \( (n - t) \) processes invoke the \( k \)-SA operation \( \text{propose}_k() \). This algorithm is very close to Algorithm 1. Its main difference lies in the replacement of \((t + 1)\) by \((n - t)\) at line 2.

```
operation propose\_2t\_n\_2(v) is
(1) view\_i ← IMSP.write\_snapshot(v); VIEW[i] ← view\_i;
(2) wait(|\{ j such that VIEW[j] \neq ⊥\}| = n - t);
(3) let view be the smallest of the previous \((n - t)\) views;
(4) return(smallest proposed value in view)
end operation.
```

Algorithm 3: Solving \((2t - n + 2)\)-set agreement in \( \mathcal{CARW}_{n,t}[n/2 \leq t < n - 1, t \text{-IS}] \)

(code for \( p_i \))

**Theorem 5.** Algorithm 3 reduces \((2t - n + 2)\)-set agreement to \( t \)-IS in \( \mathcal{CARW}_{n,t}[n/2 \leq t < n - 1] \).

**Proof**  Let \( k = 2t - n + 2 \).

Let us first consider the \( k \)-SA Termination property. There are at least \((n - t)\) correct processes, and each of them first invokes \( \text{IMSP.write\_snapshot()} \) and then writes the view it obtained in the shared array \( \text{VIEW} \) (line 1). Hence, at least \((n - t)\) entries of \( \text{VIEW} \) are eventually different from \( ⊥ \), from which follows that no process can block forever at line 2.
Let us now consider the $k$-SA Validity property. It follows from the Containment property of the $t$-IS object that any set of views deposited in $VIEW$ is not empty. Therefore, the view selected by a process at line 3 is not empty. As a view can only contain pairs, each including a proposed value (line 1), the $k$-SA Validity property follows.

Let us finally consider the $k$-SA Agreement property. Let us first observe that, due to the $t$-IS Containment property and Theorem 2, at most $n - (n - t) = t + 1$ different views can be written in the array $VIEW[1..n]$. Let $V(1)$ the smallest of these views (which contains $\ell \geq n - t$ pairs), $V(2)$ the second smallest, etc., until $V(t + 1)$ the greatest one. There are two cases according to the $(n - t)$ non-$\perp$ views obtained by a process $p_i$ at line 2. Let us remind that, as $n \leq 2t$, we have $n - t \leq t$.

- **Case 1.** The view $V(1)$ belongs to the $(n - t)$ views obtained by $p_i$. In this case, $p_i$ selects $V(1)$ at line 3 and decides at line 4 the smallest proposed value contained in $V(1)$.

- **Case 2.** The view $V(1)$ does not belong to the $(n - t)$ views obtained by $p_i$. Hence, the $(n - t)$ views obtained by any process of Case 2 belong to $\{V(2), \ldots, V(t+1)\}$. It follows that the $m = (n - t) - 1$ biggest views in $\{V(2), \ldots, V(t+1)\}$ will never be selected be the processes that are in Case 2, and consequently the set of these processes obtain at most $t - m = t - ((n - t) - 1) = 2t - n + 1$ different smallest views. Hence, these processes may decide at most $2t - n + 1$ different values at line 4.

When combining the two cases, at most $k = 2t - n + 2$ different values can be decided, which concludes the proof of the theorem.

**Corollary 3.** Implementing a $t$-IS object in $CARW_{n,t}[n/2 \leq t < n - 1]$ is impossible.

**Proof** As $t \leq n - 2$, we have $2t - n + 2 \leq t$. The proof is an immediate consequence of Theorem 5, and the fact that $(2t - n + 2)$-set agreement cannot be solved in $CARW_{n,t}[n/2 \leq t < n - 1]$ [5,21,33].

**Theorem 6.** There is no $t$-resilient consensus algorithm using $t$-immediate snapshot in $CARW_{n,t}[n/2 \leq t < n - 1]$.

The proof of the theorem is by contradiction. It assume that there is a $t$-resilient consensus algorithm $A$ for a set of processes $\{p_1, \ldots, p_n\}$, which uses a $t$-immediate snapshot object in a system where $n = 2t$ (the cases for the other values of $t$ can easily be reduced to this case).

The contradiction is obtained by simulating $A$ with two processes $Q_0$ and $Q_1$, such that $Q_0$ and $Q_1$ solve consensus despite the possible crash of one of them. As there is no wait-free consensus algorithm for 2 processes, it follows that such a consensus algorithm $A$ based on $t$-immediate snapshot objects does not exist. The proof can be found in [12].

\[ \square \]
7 Conclusion

This paper addressed the design of \( t \)-tolerant algorithms building a \( t \)-immediate snapshot (\( t \)-IS) object. Such an object in an immediate snapshot object (defined by Termination, Self-inclusion, Containment, and Immediacy properties), in a \( t \)-crash asynchronous system. Hence, it is required that each set returned to a process contains at least \( (n - t) \) pairs. Immediate snapshot corresponds to \( (n - 1) \)-immediate snapshot.

<table>
<thead>
<tr>
<th>( 1 \leq t &lt; n/2 )</th>
<th>( n/2 \leq t &lt; n - 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )-IS implements ( t )-CONS (Th. 3)</td>
<td>( t )-IS implements ( (2t - n + 2) )-Set agreement (Th. 5)</td>
</tr>
<tr>
<td>( t )-CONS implements ( t )-IS (Th. 4)</td>
<td>( t )-CONS implements ( t )-IS (Th. 4)</td>
</tr>
</tbody>
</table>

Table 1. Summary of results presented in the paper

The paper has shown that, while it is possible to build an \( (n - 1) \)-IS object in the asynchronous read/write \( (n - 1) \)-crash model, it is impossible to build a \( t \)-IS object in an asynchronous read/write \( t \)-crash model when \( 0 < t < n - 1 \). It follows that the notion of an IIS distributed model seems inoperative for these values of \( t \). The results of the paper are summarized in Table 1 where \( t \)-CONS denotes the consensus in the presence of up to \( t \) process crashes.

Interestingly, this study shows that there are two contrasting impossibility results in asynchronous read/write \( t \)-crash \( n \)-process systems. Consensus is impossible as soon as \( t > 0 \), while \( t \)-immediate snapshot is impossible as soon as \( t < n - 1 \).

As a final remark, some computability problems remain open. As an example, is it possible to implement a \( t \)-IS object from \( (2t - n + 2) \)-Set agreement?

Acknowledgments

The authors want to thank the referees for their constructive comments. This work has been partially supported by the French ANR project DISPLEXITY devoted to the study of Computability and Complexity in distributed computing, and the UNAM-PAPIIT project IN107714.

References


12. Delporte C., Fauchon H., Rajsbaum S., and Raynal M., t-Resilient immediate snapshot is impossible. Tech Report 2036, IRISA, Université de Rennes (F), 2016 https://hal.inria.fr/hal-01313556.


