Revisiting Immediate Snapshot

Carole Delporte[1], Hugues Fauconnier[1], Sergio Rajsbaum[2], Michel Raynal[3]

[1] IRIF, Université Paris Diderot, Paris, France
[2] Instituto de Matemáticas, UNAM, México D.F, 04510, México
[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 $t < n-1$ 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[1], which execute a sequence of asyn-

[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_snapshot(), that each process may invoke only once. The invocation $IMSP$.write_snapshot(v) by a process p_i adds the pair $\langle i, v\rangle$ to $IMSP$ and returns a set of pairs belonging to $IMSP$ such that the sets returned to the processes that invoke write_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 Ω (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-Writer 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 $\mathcal{CARW}_{n,t}[\emptyset]$ (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 $\mathcal{CARW}_{n,t}[t < x]$. Hence, as we assume at least one process does not crash, $\mathcal{CARW}_{n,t}[t < n]$ is a synonym of $\mathcal{CARW}_{n,t}[\emptyset]$, which (as always indicated) is called *wait-free* model. When considering t-crash models, $\mathcal{CARW}_{n,t}[t \leq \alpha]$ is less constrained than $\mathcal{CARW}_{n,t}[t < \alpha - 1]$.

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 (write() and snapshot()) which can be invoked separately by a process (usually write() before snapshot()), a immediate snapshot provides the processes with a single operation write_snapshot(). One-shot means that a process may invoke 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 $IMSP$.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 $view_i$, such that the sets returned to the processes collectively satisfy the following properties.

- Termination. The invocation of write_snapshot() by a correct process terminates.
- Self-inclusion. $\forall\, i : \langle i, v\rangle \in view_i$.
- Validity. $\forall\, i : (\langle j, v\rangle \in view_i) \Rightarrow p_j$ invoked write_snapshot(v).
- Containment. $\forall\, i, j : (view_i \subseteq view_j) \vee (view_j \subseteq view_i)$.
- Immediacy. $\forall\, i, j : (\langle i, v\rangle \in view_j) \Rightarrow (view_i \subseteq view_j)$.

It is relatively easy to show that the Immediacy property can be re-stated as follows: $\forall\, i, j : \big((\langle i, -\rangle \in view_j) \wedge (\langle j, -\rangle \in view_i)\big) \Rightarrow (view_i = view_j)$.

Implementation
Implementations of an IS object in the wait-free model $\mathcal{CARW}_{n,t}[0 < t < n]$ 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.

$r_i \leftarrow 0$; $\ell s_i \leftarrow$ initial local state of p_i (including its input, if any);
repeat forever % asynchronous IS-based rounds
  $r_i \leftarrow r_i + 1$;
  $view_i \leftarrow IMSP[r_i]$.write_snapshot$(\ell s_i)$;
  computation of a new local state ℓs_i (which contains $view_i$)
end repeat.

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 propose$_k()$. This operation allows the invoking process p_i to propose a value it passes as an input parameter (called *proposed* value), and obtain a value (called *decided* value). The object is defined by the following set of properties.

- Termination. The invocation of 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 $view$ obtained by a process is such that $|view| \geq n - k$.

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. $\square_{Theorem\ 1}$

Considering the system model $\mathcal{CARW}_{n,t}[0 < t < n-1]$, the next theorem characterizes the power of a t-IS object in term of the Containment property.

Theorem 2. *Considering the system model $\mathcal{CARW}_{n,t}[0 < t < n-1]$, and a t-IS object, let us assume that all correct processes invoke* 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 ℓ ($\ell \geq n-t$), 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* write_snapshot().

Proof It follows from the Output size property of the t-IS object that no view contains less than $(n-t)$ pairs. Let $view$ be the smallest view returned by a process, and let $\ell = |view|$. We have $\ell \geq n-t$. Moreover, due to (a) the Immediacy property (namely $(\langle i, - \rangle \in view) \Rightarrow (view_i \subseteq view)$) and (b) the minimality of $view$, it follows that $view_i = view$. As this is true for each process whose pair participates in $view$, and $\ell = |view|$, 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 $view$, or crashed during its invocation of write_snapshot(). Due to the Containment property, the others processes crash or obtain views which strictly include $view$. $\square_{Theorem\ 2}$

3 t-Immediate Snapshot is Impossible in $\mathcal{CARW}_{n,t}[0 < t < n/2]$

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

From t-IS to consensus in $\mathcal{CARW}_{n,t}[0 < t < n/2]$
Algorithm 1 reduces consensus to t-IS in the system model $\mathcal{CARW}_{n,t}[0 < t < n/2]$. As at most $t < n/2$ process may crash, at least $n - t > n/2t$ processes invoke the consensus operation $\text{propose}_1()$.

```
operation propose1(v) is
(1) view_i ← IMSP.write_snapshot(v); VIEW[i] ← view_i;
(2) wait(|{ j such that VIEW[j] ≠ ⊥}| = t + 1);
(3) let view be the smallest of the previous (t + 1) views;
(4) return(smallest proposed value in view)
end operation.
```

Algorithm 1: Solving consensus in $\mathcal{CARW}_{n,t}[0 < t < n/2, t\text{-IS}]$ (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 $[\perp, \cdots, \perp]$. The aim of $VIEW[i]$ is to store the view obtained by p_i from the t-IS object $IMSP$.

When it calls $\mathsf{propose}_1(v)$, a process p_i invokes first the t-IS object, in which it deposits the pair $\langle i, v\rangle$, 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* $\mathcal{CARW}_{n,t}[0 < t < n/2]$.

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 $\perp$. 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_view (or crashed before returning from $\mathsf{write\_snapshot}()$), and this view is the smallest view obtained by a process and its size is $|min\_view| = \ell$. 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_view, and consequently we have $view = min\_view$ 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. $\square_{Theorem\ 3}$

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

Proof The proof is an immediate consequence of Lemma 3, and the fact that consensus cannot be solved in $\mathcal{CARW}_{n,t}[0 < t < n/2]$ [25]. $\square_{Corollary\ 1}$

4 From Consensus to t-IS in $\mathcal{CARW}_{n,t}[0 < t \leq n - 1]$

Algorithm *2* describes a reduction of t-IS to consensus in $\mathcal{CARW}_{n,t}[0 < t \leq n - 1]$. 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 $\mathsf{write\_snapshot}(v_i)$ 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 $\mathsf{return}()$ statement at line 9 terminates the $\mathsf{write\_snapshot}()$ operation invoked by p_i. The termination of $T2$ does not kill the task $T1$ which may continue executing.

```
operation write_snapshot(v_i) is
(1)  REG[i] ← v_i; view_i ← ∅; dec_i ← ∅; k ← −1; launch the tasks T1 and T2.

(2)  task T1 is
(3)    repeat k ← k + 1;
(4)          wait(∃ a set aux_i: (dec_i ⊂ aux_i) ∧ (|aux_i| = n − t + k)
                    ∧ (aux_i ⊆ {⟨j, REG[j]⟩ such that REG[j] ≠ ⊥}));
(5)          dec_i ← CONS[n − t + k].propose_1(aux_i);
(6)          if (⟨i, v_i⟩ ∈ dec_i) ∧ (view_i = ∅) then view_i ← dec_i end if
(7)    until (k = t) end repeat
(8)  end task T1.

(9)  task T2 is wait(view_i ≠ ∅); return(view_i) end task T2.
end operation.
```

Algorithm 2: Implementing t-IS in $\mathcal{CARW}_{n,t}[0 < t < n/2, \text{CONS}]$ (code for p_i)

Task $T1$ (lines 2-8) has two aims: provide p_i with a view $view_i$ (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 aux_i, which (a) contains $(n - t + k)$ pairs, (b) contains the set set of pairs dec_i decided during the previous iteration, and (c) contains only pairs extracted from the array $REG[1..n]$. This is captured by the predicate of line 4.
- Then, p_i proposes the set aux_i to the consensus object $CONS[n-t+k]$ associated with the current iteration step (line 5). The set decided is stored in dec_i.
- Finally, if its pair $\langle i, v_i\rangle$ belongs to dec_i and p_i has not yet decided (i.e., no set has yet been assigned to $view_i$), it does it by writing dec_i in $view_i$. 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* $\mathcal{CARW}_{n,t}[0 < t \leq n - 1]$.

Proof The Self-inclusion property follows directly from the predicate $\langle i, v_i\rangle \in dec_i$ used before assigning dec_i to $view_i$ 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 $REG[i]$, (b) this atomic variable is written at most once (line 1), and (c) the predicate $REG[j] \neq \perp$ is used at line 4 to extract values from $REG[1..n]$.

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); k_j, dec_j, and $view_j$ being defined similarly, we also have $\langle j, v_j\rangle \in dec_j[k_j] = view_j$. As by assumption we have $\langle j, v_j\rangle \in view_i$ and $\langle i, v_i\rangle \in view_j$, we also have $\{\langle i, v_i\rangle, \langle j, v_j\rangle\} \subseteq dec_i[k_i] = view_i$ and $\{\langle i, v_i\rangle, \langle j, v_j\rangle\} \subseteq dec_j[k_j] = view_j$. Due to the Agreement property of the consensus objects, we have $dec_i[k_i] = dec_j[k_i]$, and $dec_i[k_j] = dec_j[k_j]$.

Let us assume that $k_i < k_j$. This is not possible because, on the one side, $\langle j, v_j\rangle \in dec_i[k_i] = dec_j[k_i]$, and, on the other side, k_j is the only iteration step at which we have $\langle j, v_j\rangle \in dec_j \wedge view_j = \emptyset$ (and consequently $view_j$ is assigned the value in $dec_j[k_j]$). For the same reason, we cannot have $k_i > k_j$. It follows that $k_i = k_j$. Hence, as $dec_i[k_i] = dec_j[k_i]$, p_i and p_j obtain the very same view (and this occurs during the same iteration step).

As far as the Containment property is concerned, we have the following. Considering the iteration number k, let us first observe that, due to the predicate $|aux_i| = n-t+k$ (line 4), the set output by $CONS[n-t+k]$ contains $n-t+k$ pairs. Hence, the sequence of consensus outputs sets whose size is increased by 1 at each instance. Let us now observe that, due to the predicate $dec_i \subset aux_i$ (line 4), the set output by $CONS[n-t+k+1]$ is a superset of the set output by the previous consensus instance $CONS[n-t+k]$. It follows that the sequence of pairs output by the consensus instances is such that each set of pairs includes the previous set plus one new element, from which the Containment property follows.

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 $CONS[n-t+k].\mathsf{propose}_1()$ (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 $(n-t)$, $(n-t+1)$, ..., 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 dec is such that, while $view_i = \emptyset$, we have $\langle i, v_i \rangle \in dec$, which concludes the proof of the Termination property. $\square_{Theorem\ 4}$

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). $\square_{Theorem\ 2}$

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 $\mathcal{CARW}_{n,t}[n/2 \leq t < n-1, t$-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 $\mathsf{propose}_k()$. 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 $\mathsf{propose}_{2t-n+2}(v)$ **is**
(1) $view_i \leftarrow IMSP.\mathsf{write\_snapshot}(v)$; $VIEW[i] \leftarrow view_i$;
(2) $\mathsf{wait}(|\{ j \text{ such that } VIEW[j] \neq \perp\}| = n-t)$;
(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 $\mathcal{CARW}_{n,t}[n/2 \leq t < n-1, t$-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 $IMSP.\mathsf{write\_snapshot}()$ and then writes the view it obtained in the shared array $VIEW$ (line 1). Hence, at least $(n-t)$ entries of $VIEW$ are eventually different from $\perp$, 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 $n-(n-t)+1 = t+1$ 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 $\{V(2), \cdots, V(t+1)\}$. It follows that the $m = (n-t)-1$ biggest views in $\{V(2), \cdots, V(t+1)\}$ 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. $\square_{Theorem\ 5}$

Corollary 3. *Implementing a t-IS object in $\mathcal{CARW}_{n,t}[n/2 \leq t < n-1]$ 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 $\mathcal{CARW}_{n,t}[n/2 \leq t < n-1]$ [5,21,33]. $\square_{Corollary\ 3}$

6 t-Immediate Snapshot and Consensus in $\mathcal{CARW}_{n,t}[n/2 \leq t < n-1]$

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

The proof the theorem is by contradiction. It assume that there is a t-resilient consensus algorithm $\mathcal{A}$ for a set of processes $\{p_1, \cdots, p_n\}$, 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 $\mathcal{A}$ 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 $\mathcal{A}$ based on t-immediate snapshot objects does not exist. The proof can be found in [12].

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.

$1 \leq t < n/2$	$n/2 \leq t < n-1$
t-IS implements t-CONS (Th. 3)	t-IS implements $(2t-n+2)$-Set agreement (Th. 5) t-IS does not implement t-CONS (Th.6)
t-CONS implements t-IS (Th. 4)	t-CONS implements t-IS (Th. 4)

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

1. Afek Y., Attiya H., Dolev D., Gafni E., Merritt M. and Shavit N., Atomic snapshots of shared memory. *Journal of the ACM*, 40(4):873-890 (1993)
2. Anderson J., Multi-writer composite registers. *Distributed Computing*, 7(4):175-195 (1994)
3. Attiya H., Bar-Noy A., Dolev D., Peleg D., and Reischuk R., Renaming in an asynchronous environment. *Journal of the ACM*, 37(3):524-548 (1990)
4. Attiya H. and Welch J., *Distributed computing: fundamentals, simulations and advanced topics*, (2d Edition), Wiley-Interscience, 414 pages (2004)

5. Borowsky E. and Gafni E., Immediate atomic snapshots and fast renaming. *Proc. 12th ACM Symposium on Principles of Distributed Computing (PODC'93)*, pp. 41-50 (1993)
6. Borowsky E. and Gafni E., Generalized FLP impossibility results for *t*resilient asynchronous computations. *Proc. 25th ACM Symposium on Theory of Computation (STOC'93)*, California (USA), pp. 91-100 (1993)
7. Borowsky E. and Gafni E., A simple algorithmically reasoned characterization of wait-free computations. *Proc. 16th ACM Symposium on Principles of Distributed Computing (PODC'97)*, ACM Press, pp. 189-198 (1997)
8. Borowsky E., Gafni E., Lynch N. and Rajsbaum S., The BG distributed simulation algorithm. *Distributed Computing*, 14:127-146 (2001)
9. Castañeda A., Rajsbaum S., and Raynal M., Specifying concurrent problems: beyond linearizability and up to tasks. *Proc. 29th Symposium on Distributed Computing (DISC'15)*, Springer LNCS 9363, pp. 420-435 (2015)
10. Chandra T., Hadzilacos V., and Toueg S., The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685-722 (1996)
11. Chaudhuri S., More choices allow more faults: set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132-158 (1993)
12. Delporte C., Fauconnier 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.
13. Fischer M.J., Lynch N.A., and Paterson M.S., Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374-382 (1985)
14. Gafni E., Kuznetsov P., and Manolescu C., A generalized asynchronous computability theorem. *Proc. 33th ACM Symposium on Principles of Distributed Computing (PODC'94)*, ACM Press, paP. 222-231 (2014)
15. Gafni E. and Rajsbaum S., Recursion in distributed computing. *Proc. 12th Int'l Conference on Stabilization, Safety, and Security of Distributed Systems (SSS'10)*, Springer LNCS 6366, pp. 362-376 (2010)
16. Herlihy M. P., Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124-149 (1991)
17. Herlihy M.P., Kozlov D., and Rajsbaum S., *Distributed computing through combinatorial topology*, Morgan Kaufmann/Elsevier, 336 pages, ISBN 9780124045781 (2014)
18. Herlihy M.P., Luchangco V., and Moir M., Obstruction-free synchronization: double-ended queues as an example. *Proc. 23th Int'l IEEE Conference on Distributed Computing Systems (ICDCS'03)*, IEEE Press, pp. 522-529, 2003.
19. Herlihy M., Rajsbaum S., and Raynal M., Power and limits of distributed computing shared memory models. *Theoretical Computer Science*, 509:3-24 (2013)
20. Herlihy M. P. and Shavit, N., The asynchronous computability theorem for *t*-resilient tasks. *Proc. 25th ACM Symposium on Theory of Computing (STOC 1993)*, ACM Press, pp. 111-120 (1994)
21. Herlihy M. P. and Shavit, N., The topological structure of asynchronous computability. *Journal of the ACM*, 46(6):858-923 (1999)
22. Herlihy M. P. and Wing J. M., Linearizability: a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463-492 (1990)
23. Lamport L., On interprocess communication, Part I: basic formalism. *Distributed Computing*, 1(2):77-85 (1986)
24. Lo W.-K. and Hadzilacos V., Using failure detectors to solve consensus in asynchronous shared memory systems. *Proc. 8th Int'l Workshop on Distributed Algorithms (WDAG'94)*, Springer LNCS 857, pp. 280-295 (1994)
25. Loui M. and Abu-Amara H., Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, 4:163-183, JAI Press (1987)

26. Neiger G., Set-linearizability. Brief announcement in *Proc. 13th ACM Symposium on Principles of Distributed Computing (PODC'94)*, ACM Press, page 396 (1994)
27. Rajsbaum S., Iterated shared memory models. *Proc. 9th Latin American Symposium on Theoretical Informatics (LATIN'10)*, Springer LNCS 6034, pp. 407-416 (2010)
28. Rajsbaum, S. and Raynal, M., An introductory tutorial to concurrency-related distributed recursion. *Bulletin of the European Association of TCS*, 111:57-75 (2013)
29. Rajsbaum S., Raynal M., and Travers C., The iterated restricted immediate snapshot model. *Proc. 14th Annual Int'l Conference on Computing and Combinatorics (COCOON'08)*, Springer LNCS 5092, pp. 487-497 (2008)
30. Rajsbaum, S., Raynal, M., and Travers, C., An impossibility about failure detectors in the iterated immediate snapshot model. *Information Processing Letters*, 108(3):160-164 (2008)
31. Raynal M., *Concurrent programming: algorithms, principles and foundations*. Springer, 515 pages, ISBN 978-3-642-32026-2 (2013)
32. Raynal M. and Stainer J., Increasing the power of the iterated immediate snapshot model with failure detectors. *Proc. 19th Int'l Colloquium on Structural Information and Communication Complexity (SIROCCO'12)*, Springer LNCS 7355, pp. 231-242 (2012)
33. Saks M. and Zaharoglou F., Wait-free k-set agreement is impossible: the topology of public knowledge. *SIAM Journal on Computing*, 29(5):1449-1483 (2000)
34. Taubenfeld G., Synchronization algorithms and concurrent programming. *Pearson Prentice-Hall*, 423 pages, ISBN 0-131-97259-6 (2006)