t-résilient snapshot immédiat
Carole Delporte-Gallet, Hugues Fauconnier, Sergio Rajsbaum, Michel Raynal

To cite this version:
Carole Delporte-Gallet, Hugues Fauconnier, Sergio Rajsbaum, Michel Raynal. t-résilient snapshot immédiat. ALGOTEL 2017 - 19èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications, May 2017, Quiberon, France. Aspects Algorithmiques des Télécommunications. <hal-01517320>
Dans un système de \( n \) processus communiquant par mémoire partagée, un snapshot immédiat est un état de la mémoire assurant que si l’écriture réalisée par \( q \) est dans le snapshot de \( p \) alors le snapshot de \( p \) contient le snapshot obtenu par \( q \). Un snapshot immédiat peut être réalisé dans un système où au plus \( n - 1 \) processus peuvent tomber en panne. Dans un système où \( t \) processus peuvent tomber en panne, un processus peut obtenir un snapshot contenant les valeurs écrites par jusqu’à \( n - t \) processus. On définit ainsi le \( t \)-résilient snapshot immédiat et on établit des liens entre ce problème et les problèmes d’accord comme le consensus et le \( k \)-accord.

**Mots-clés :** Mémoire partagé, snapshot immédiat, tolérance aux pannes, consensus, \( k \)-accord.

1 Introduction

We consider a distributed computing model of \( n \) asynchronous processes among which any subset of up to \( t \) processes may crash communicating by shared memory.

The snapshot object was first proposed over a decade ago [1, 2] and has since been intensively studied by the distributed algorithms community. A snapshot object can be seen as a data structure initially empty, 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 two operations denoted update and snap. The invocation \( \text{update} (v) \) by a process \( p_i \) adds the pair \( (i, v) \) to the data structure and the operation \( \text{snap} \) returns all the pairs already written in the data structure.

The immediate snapshot (IS) object has been introduced in [3, 11], and later investigated in [5, 15]. It is a variant of the snapshot object. An immediate snapshot provides the processes with a single operation \( \text{write\_snapshot}() \) that a process may invoke at most once. The invocation \( \text{write\_snapshot}(v) \) by a process \( p_i \) adds the pair \( (i, v) \) to the object and returns a set of pairs \( \text{view}_i \) belonging to the object such that if \( (j, w) \) is in \( \text{view}_j \) then \( \text{view}_j \subseteq \text{view}_i \).

The noteworthy feature of the iterated immediate snapshot model is the following. It has been shown by Borowsky and Gafni in [5], 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 [14]. It is also the basis of the combinatorial topology approach for distributed computing (e.g., [10]). Hence, IS objects constitute the algorithmic foundation of distributed iterated computing models.

When considering the \( t \)-crash \( n \)-process model where \( t < n \), 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 \( t \)-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 \( \text{write\_snapshot}() \) by a process returns a set including at least \((n - t)\) pairs.

\[\text{t-résilient snapshot immédiat}^{†}\]

Carole Delporte-Gallet\(^1\)\(^‡\) et Hugues Fauconnier\(^1\) et Sergio Rajsbaum\(^2\) et Michel Raynal\(^3\)

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

\(^†\)This paper is an extended abstract of [7]; the original title is \( t \)-resilient snapshot immédiat.

\(^‡\)Carole Delporte-Gallet, Hugues Fauconnier and Michel Raynal are supported by ANR DESCARTES
(while it would return a set of \( x \) pairs with \( 1 \leq x \leq n \) if the object was an immediate snapshot 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.

Implementations of an \((n - 1)\)-immediate snapshot object is described in [3]. For the other values of \( t \) (\( 0 < t < n - 1 \)), 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\(^\dagger\) 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. First, an IS object is a snapshot object (a) whose operations update and snap are glued together in a single operation \texttt{write\_snapshot()}\(^\dagger\), and (b) satisfying an additional property linking the sets of pairs returned by concurrent invocations. Then, as already indicated, a \( t \)-IS object is an IS object such that the sets returned by \texttt{write\_snapshot()} contain at least \((n - t)\) pairs. This property on the sets returned by a snapshot object can be trivially implemented in a \( t \)-crash \( n \)-process model. Hence, while a \( t \)-snapshot object can be implemented in the \( t \)-crash \( n \)-process model, a \( t \)-IS object cannot when \( 0 < t < n - 1 \). Second, because in general smallest is the number of possible faults easiest is the implementation. But in the \( t \)-IS, a small number of failure, induces a harder problem each output set must have the size at least \( n - t \).

2 Model

We consider a distributed computing model of \( n \geq 3 \) asynchronous sequential processes denoted \( p_1, \ldots, p_n \) among which any subset of up to \( t \) (\( 0 < t < n \)) processes may crash. The processes cooperate by reading and writing Single-Writer Multi-Reader atomic read/write registers [13].

One-shot immediate snapshot object. An immediate snapshot object (IS) 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 \texttt{write\_snapshot(\texttt{v})}. This invocation adds the pair \((i, v)\) to the object, and returns to \( p_i \) a set, called view and denoted \( \texttt{view}_i \), such that the sets returned to the processes collectively satisfy the following properties.

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

\( k \)-Set agreement. \( k \)-Set agreement was introduced by S. Chaudhuri [6], 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 \texttt{propose}(\texttt{v})\(^\dagger\). This operation allows the invoking process \( p_i \) to propose a value it passes as an input parameter (called \texttt{proposed} value), and obtain a value (called \texttt{decided} value). The object is defined by the following set of properties.

- Termination. The invocation of \texttt{propose}(\texttt{v}) 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 [8, 4, 11, 16] that the problem is impossible to solve if \( k \leq t \).

\( t \)-Immediate Snapshot. A \( t \)-immediate snapshot object (denoted by \( t \)-IS) is an immediate snapshot object with the following additional property.

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

3 Results

\( t \)-Immediate Snapshot is Impossible if \( 0 < t < n - 1 \). This section presents an algorithm Figure 1 to achieve (1) consensus from \( t \)-IS for \( 0 < t < n/2 \), and (2) \( k \)-Set agreement (in short \( k \)-SA) from \( t \)-IS for

\( A \) is equivalent to \( B \) if \( A \) can be (computationally) reduced to \( B \) and reciprocally.
$t$-resilient immediate snapshot

```plaintext
1 PROPOSE(v)
2 begin
3 view ← I.write_snapshot(v); /* I shared immediate snapshot*/
4 VIEW[i] ← view; /* VIEW is a shared array*/
5 wait(|{ j such that VIEW[j] ≠ ⊥}| = t + 1);
6 let view be the smallest of the previous $(t + 1)$ views;
7 return(smallest proposed value in view)
8 end PROPOSE
```

**Figure 1:** Solving consensus from $t$-IS if $0 < t < n/2$(code for $p_1$)

```plaintext
1 write_snapshot(v);
2 begin
3 S.update((i, vi)); /* S shared snapshot*/
4 view ← θ; dec ← θ; k ← −1; launch the tasks T1 and T2.
5 Task T1 :
6 repeat k ← k + 1
7 do aux ← S.snapshot()
8 until (dec ⊆ aux ∧ |aux| ≥ n − t)
9 dec ← CONS[k], propose$(i)$
10 if ((i, vi) ∈ dec) ∧ (view = θ) then view ← dec end if
11 until |aux| = n
12 end task T1
13 Task T2 : wait(view θ); return(view) end task T2.
14 end write_snapshot
```

**Figure 2:** Implementing $t$-IS with consensus (code for $p_0$)

$k = 2t − n + 2$ (e.g., $(n − 2)$-SA agreement from $(n − 2)$-IS if $t = n − 2$). As these problems are impossible to solve [4, 11, 16], we show that it is impossible to implement a $t$-IS object when $0 < t < n − 1$.

Intuitively this algorithm works because there is a set of at least $ℓ ≥ n − t$ processes, that obtained the same view $min_view$ (or crashed before returning from write_snapshot()), and this view is the smallest view obtained by a process and its size is $|min_view| = ℓ$. If $0 < t < n/2$, as $ℓ ≥ n − t$ and $(n − t) + (t + 1) > n$, it follows from the waiting predicate, that, any process that executes line 5, obtains a copy of $min_view$, and consequently we have $view = min_view$ at line 5. It follows that no two processes can decide different values. If $n/2 ≤ t < n − 1$, we have $n − t ≤ t$. The $m = (n − t) − 1$ biggest views will never be selected by the processes, and consequently 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.

**Theorem 1** A $t$-IS object cannot be implemented if $0 < t < n − 1$.

**From Consensus to $t$-IS if $0 < t ≤ n − 1$**. While a snapshot object is atomic (operations on a snapshot object can be linearized [12]), an IS object and a $k$-immediate snapshot objects are not atomic (its operations cannot always be linearized).

So we cannot apply the universality result of Herlihy [9], and we have to write a specific algorithm given Figure 2.

**$t$-Immediate Snapshot and $k$-Set agreement.** With the algorithms Figures 1 and 2, we get:

**Theorem 2** Consensus and $t$-IS are equivalent if $0 < t < n/2$.

We have shown that from $t$-IS when $n/2 ≤ t < n − 2$ we can implement $(2t − n + 2)$-Set agreement. Can we do consensus as in the case $0 < t < n/2$? By a simulation argument, we show that consensus is not solvable with $t$-immediate snapshot when $n/2 ≤ t < n$ 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 $n/2 ≤ t < n$. 


Theorem 3 If $0 < t < n/2$ then $t$-IS can implement $(2t - n + 2)$-Set agreement and cannot implement consensus. Consensus implements $t$-IS.

<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 and consensus are equivalent</td>
<td>$t$-IS implements $(2t - n + 2)$-Set agreement</td>
</tr>
<tr>
<td>$t$-IS does not implement consensus</td>
<td>consensus implements $t$-IS</td>
</tr>
</tbody>
</table>

**Table 1:** Summary of results presented in the paper

4 Conclusion

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 Iterative immediate snapshot distributed model seems inoperative for these values of $t$. The results of the paper are summarized in Table 1.

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?

Références


