Improvements to a Roll-Back Mechanism for Asynchronous Checkpointing and Recovery

Monika Kapus-Kolar
Department of Communication Systems, Jožef Stefan Institute
Jamova cesta 39, 1000 Ljubljana, Slovenia
E-mail: monika.kapus-kolar@ijs.si

Keywords: asynchronous checkpointing, recovery, maximum consistent state

Received: September 19, 2007

We indicate the existence of a logical flaw in a recently published recovery algorithm for distributed systems and suggest a correction. We also improve the associated communication protocol, the prevention of multiple concurrent instantiations of the algorithm, the handling of obsolete messages and the organization of the stable storing of the relevant data.

Povzetek: Opozarjamo na logično napako v pred kratkim objavljenem algoritmu za okrevanje v porazdeljenih sistemih in predlagamo popravek.

1 Introduction

Gupta, Rahimi and Yang recently proposed a novel recovery algorithm for distributed systems in which checkpoints are taken asynchronously [1]. A checkpoint taken by a process is a snapshot of its local state, stored in a stable storage, so that the process can roll back to it, if this becomes necessary. The start of a process is also one of its checkpoints. Asynchronous checkpointing means that processes take their checkpoints independently.

A failure in a distributed system in principle requires that all its constituent processes roll back, to a global state from which the system can resume its operation as if it had started from it, i.e., to a *globally consistent set of local checkpoints* (GCSLC), ideally to the so called *maximum GCSLC*, in which every local checkpoint is as recent as possible. In the case of asynchronous checkpointing, when a failure occurs, the processes have yet to find the maximum GCSLC. They do that by running a *checkpoint coordination algorithm* (CCA).

Alternatively, the processes might agree to restart from the GCSLC which they currently treat as the starting state of the system, i.e., from the current *recovery line*. This is the most recently computed maximum GCSLC, initially the actual starting state of the system. [1] suggests that the processes occasionally initiate a CCA just for advancing the recovery line.

In this paper, we demonstrate that, because of a subtle logical flaw, the CCA of [1] sometimes returns a checkpoint set which is *not* globally consistent. We correct the flaw and also suggest some other improvements. The rest of the paper is organized as follows. In the next section, we describe the system and the testing of checkpoint consistency and give a brief outline of the CCA of [1]. In Section 3, we explain and correct the flaw. In Section 4, we suggest several other improvements of the CCA. A detailed specification of the improved CCA is given in Section 5. Section 6 suggests that checkpointing and recovery-line advancing in the absence of failures should be more flexible. As the proposed CCA correction increases the usage of the local stable storages, we in Section 7 suggest how to organize them. Section 8 comprises a discussion and conclusions.

2 Preliminaries

2.1 System properties

Of the assumptions explicitly or implicitly stated in [1] for the distributed system considered, the following seem important:

- The set of the constituent processes is a fixed $\{P_j | j \in N\}$ with N a $\{1, \ldots, n\}$. Every process is virtually always aware of the global time.
- From every process P_j to every other process $P_{j'}$, there is virtually a reliable first-in-first-out channel with the worst-case transit delay not exceeding a predefined $T_{j,j'}$. The channels are the only means of inter-process communication. Processes exchange *application messages* (AMs) and *CCA messages* (CMs).
- When a process fails, no other process fails simultaneously and the system does not experience another failure until every process restarts.

2.2 Testing checkpoint consistency

A GCSLC is characterized by all its members being mutually consistent. A checkpoint $C_{j,r}$ of a process P_j is consistent with a checkpoint $C_{j',r'}$ of a process $P_{j'}$ if none of the processes has recorded a reception of an AM from the partner for which the partner has not recorded a transmission [1].

As channels are reliable queues, it is acceptable that processes record their AM transmissions and receptions simply by counting them, relative to the recovery line [1], for this is where the system virtually started. For a checkpoint $C_{j,r}$ and a process $P_{j'}$, let $S_{j,r,j'}$ and $R_{j,r,j'}$ indicate how many AMs P_j has sent to or, respectively, received from $P_{j'}$, where $S_{j,r,j}$ and $R_{j,r,j}$ are by definition zero.

Suppose that the currently considered checkpoint set is a $\{C_{j,r(j)}|j \in N\}$. To check that it is a GCSLC, one in principle has to check $R_{j,r(j),j'} \leq S_{j',r(j'),j}$ for every two processes P_j and $P_{j'}$. This is what the CCA of [2] does, by letting every process P_j check $\bigwedge_{j' \in (N \setminus \{j\})}(R_{j,r(j),j'} \leq S_{j',r(j'),j})$. The CCA of [1] works under the assumption that this can be done simply by letting every process P_j check $\sum_{j' \in (N \setminus \{j\})} R_{j,r(j),j'} \leq \sum_{j' \in (N \setminus \{j\})} S_{j',r(j'),j}$. In both CCAs, if the adopted test fails for a P_j, the process starts considering an earlier checkpoint, namely the most recent checkpoint $C_{j,r'(j)}$ with $\bigwedge_{j' \in (N \setminus \{j\})}(R_{j,r'(j),j'} \leq S_{j',r(j'),j})$ or $\sum_{j' \in (N \setminus \{j\})} R_{j,r'(j),j'} \leq \sum_{j' \in (N \setminus \{j\})} S_{j',r(j'),j}$, respectively.

2.3 Outline of the algorithm

In the CCA of [1], all communication goes over the initiator of the particular algorithm instance (see Example 1 in the next section). The initiator process repeatedly polls every process, virtually also itself, for the value of the transmission counters of the local candidate for a recovery-line checkpoint. Every such request carries all the information on the transmission counters of the current candidate checkpoints, if any, which the recipient needs for deciding which checkpoint to consider in the next iteration. The first candidate checkpoint of a process is always its most recent checkpoint.

After every process replies, the initiator might detect that the candidate checkpoint set has changed. In that case, it starts another iteration. Otherwise, it broadcasts an indication that a GCSLC has been found. Finally, every process promotes its currently considered checkpoint into its recovery-line checkpoint, from which it, if so requested by the initiator, subsequently restarts. The algorithm covers also the possibility that the initiator requests an immediate restart, from the current recovery line.

3 A flaw and a correction

3.1 The problem of inaccurate information

When a $\{C_{j,r(j)}|j \in N\}$ is checked for being a GCSLC, each $S_{j,r(j),j'}$ may be any natural up to and including the number of the AMs sent from P_j to $P_{j'}$, and each $R_{j,r(j),j'}$ may be any natural up to and including the number of the AMs received at P_j from $P_{j'}$, for remember that the checkpointing is entirely asynchronous. It might, hence, happen that for a process P_j, the test $\sum_{j' \in (N \setminus \{j\})} R_{j,r(j),j'} \leq \sum_{j' \in (N \setminus \{j\})} S_{j',r(j'),j}$ adopted by [1] succeeds in spite of $\bigwedge_{j' \in (N \setminus \{j\})}(R_{j,r(j),j'} \leq S_{j',r(j'),j})$ false, so that P_j fails to detect that $\{C_{j,r(j)}|j \in N\}$ is *not* a GCSLC. If the other processes also fail to detect that the checkpoint set is not consistent, the CCA terminates prematurely and the set unacceptably becomes the new recovery line.

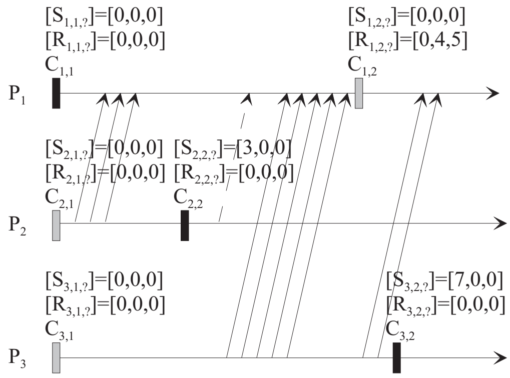


Figure 1: The situation considered in Examples 1–5. The black checkpoints represent the maximum GCSLC. The dashed message is the one making $C_{1,2}$ inconsistent with $C_{2,2}$.

Example 1. *A possible problematic scenario of a system consisting of processes P_1 to P_3 and using the CCA of [1] (see also Fig. 1):*

(1) Upon the start of the system, every process P_j takes a checkpoint $C_{j,1}$, with every $S_{j,1,j'}$ and $R_{j,1,j'}$ zero.

(2) P_2 sends to P_1 three promptly received AMs and then takes a checkpoint $C_{2,2}$, with $S_{2,2,1} = 3$ and $S_{2,2,3} = R_{2,2,1} = R_{2,2,3} = 0$.

(3) P_2 sends to P_1 another promptly received AM and P_3 sends to P_1 five promptly received AMs. P_1 then takes a checkpoint $C_{1,2}$, with $R_{1,2,2} = 4$, $R_{1,2,3} = 5$ and $S_{1,2,2} = S_{1,2,3} = 0$.

(4) P_3 sends to P_1 two more promptly received AMs and then takes a checkpoint $C_{3,2}$, with $S_{3,2,1} = 7$ and $S_{3,2,2} = R_{3,2,1} = R_{3,2,2} = 0$.

(5) P_1 undergoes a failure.

(6) After P_1 recovers, it starts considering its most recent checkpoint $C_{1,2}$ and invites P_2 and P_3 to a new instance of the CCA.

(7) In response, P_2 starts considering its most recent checkpoint $C_{2,2}$ and sends to P_1 a CM carrying $S_{2,2,1}$ and $S_{2,2,3}$, while P_3 starts considering its most recent checkpoint $C_{3,2}$ and sends to P_1 a CM carrying $S_{3,2,1}$ and $S_{3,2,2}$.

(8) When P_1 receives the responses, it sends $(S_{1,2,2} + S_{3,2,2})$ to P_2 and $(S_{1,2,3} + S_{2,2,3})$ to P_3. It also observes $(R_{1,2,2} + R_{1,2,3}) = 9 \leq (S_{2,2,1} + S_{3,2,1}) = 10$ and consequently decides to continue considering $C_{1,2}$, recording the decision by setting the local flag to 0.

(9) Upon receiving $(S_{1,2,2} + S_{3,2,2})$, P_2 observes $(R_{2,2,1}+R_{2,2,3}) = 0 \leq (S_{1,2,2}+S_{3,2,2}) = 0$ and therefore continues considering $C_{2,2}$, indicating that to P_1 by a CM carrying flag 0 and $S_{2,2,1}$ and $S_{2,2,3}$. Likewise, P_3 upon receiving $(S_{1,2,3} + S_{2,2,3})$ observes $(R_{3,2,1} + R_{3,2,2}) = 0 \leq (S_{1,2,3} + S_{2,2,3}) = 0$ and therefore continues considering $C_{3,2}$, indicating that to P_1 by a CM carrying flag 0 and $S_{3,2,1}$ and $S_{3,2,2}$.

(10) After receiving the two indications, P_1, observing that the flag of every process is 0, concludes that a GCSLC has been found, tells P_2 and P_3 to restart from $C_{2,2}$ and $C_{3,2}$, respectively, and finally restarts from $C_{1,2}$.

(11) Because of $S_{2,2,1} = 3$ before the restart, P_2 after restarting retransmits the fourth AM to P_1. Because of $R_{1,2,2} = 4$ before the restart, P_1 erroneously interprets the AM as the fifth from P_2 and consequently takes an inappropriate action.

The simplification from [1] described in Section 2.2 is, hence, unacceptable.

3.2 The Necessary Changes of the Algorithm

The indispensable changes to the CCA are the following:

- Where a process P_j originally stores in its stable storage a $\Sigma_{j'\in(N\setminus\{j\})}R_{j,r(j),j'}$, it must actually store $R_{j,r(j),j'}$ of every other process $P_{j'}$.
- Where the initiator process originally sends to a process P_j a $\Sigma_{j'\in(N\setminus\{j\})}S_{j',r(j'),j}$, it must actually send a CM containing $S_{j',r(j'),j}$ of every process $P_{j'}$ other than P_j.
- Where a process P_j originally tests a $\sum_{j'\in(N\setminus\{j\})} R_{j,r(j),j'} \leq \sum_{j'\in(N\setminus\{j\})} S_{j',r(j'),j}$, it must actually test $\bigwedge_{j'\in(N\setminus\{j\})}(R_{j,r(j),j'} \leq S_{j',r(j'),j})$.

Example 2. *If the indispensable CCA corrections are made in Example 1, the scenario after its seventh step takes the following direction (see also Fig. 1):*

(8) After P_1 receives the responses, it sends a CM carrying $S_{1,2,2}$ and $S_{3,2,2}$ to P_2 and a CM carrying $S_{1,2,3}$ and $S_{2,2,3}$ to P_3. It also observes $R_{1,2,2} = 4 > S_{2,2,1} = 3$ and consequently, because $R_{1,1,2} = 0 \leq S_{2,2,1} = 3$ and $R_{1,1,3} = 0 \leq S_{3,2,1} = 7$, starts considering $C_{1,1}$, recording the change of focus by setting the local flag to 1.

(9) Upon receiving $S_{1,2,2}$ and $S_{3,2,2}$, P_2 observes $R_{2,2,1} = 0 \leq S_{1,2,2} = 0$ and $R_{2,2,3} = 0 \leq S_{3,2,2} = 0$ and therefore continues considering $C_{2,2}$, indicating that to P_1 by a CM carrying flag 0 and $S_{2,2,1}$ and $S_{2,2,3}$. Likewise, P_3 upon receiving $S_{1,2,3}$ and $S_{2,2,3}$ observes $R_{3,2,1} = 0 \leq S_{1,2,3} = 0$ and $R_{3,2,2} = 0 \leq S_{2,2,3} = 0$ and therefore continues considering $C_{3,2}$, indicating that to P_1 by a CM carrying flag 0 and $S_{3,2,1}$ and $S_{3,2,2}$.

(10) After receiving the two indications, P_i, observing that there is a process with a non-zero flag, sends a CM carrying $S_{1,1,2}$ and $S_{3,2,2}$ to P_2 and a CM carrying $S_{1,1,3}$ and $S_{2,2,3}$ to P_3. It also observes $R_{1,1,2} = 0 \leq S_{2,2,1} = 3$ and $R_{1,1,3} = 0 \leq S_{3,2,1} = 7$ and consequently decides to continue considering $C_{1,1}$, recording the decision by setting the local flag to 0.

(11) Upon receiving $S_{1,1,2}$ and $S_{3,2,2}$, P_2 observes $R_{2,2,1} = 0 \leq S_{1,1,2} = 0$ and $R_{2,2,3} = 0 \leq S_{3,2,2} = 0$ and therefore continues considering $C_{2,2}$, indicating that to P_1 by a CM carrying flag 0 and $S_{2,2,1}$ and $S_{2,2,3}$. Likewise, P_3 upon receiving $S_{1,1,3}$ and $S_{2,2,3}$ observes $R_{3,2,1} = 0 \leq S_{1,1,3} = 0$ and $R_{3,2,2} = 0 \leq S_{2,2,3} = 0$ and therefore continues considering $C_{3,2}$, indicating that to P_1 by a CM carrying flag 0 and $S_{3,2,1}$ and $S_{3,2,2}$.

(12) After receiving the two indications, P_1, observing that the flag of every process is 0, correctly concludes that a GCSLC has been found, tells P_2 and P_3 to restart from $C_{2,2}$ and $C_{3,2}$, respectively, and finally restarts from $C_{1,1}$.

4 Optimization of process communication

If processes exchange transmission counters instead of just their sums, minimization of the number and the size of the exchanged CMs becomes even more important. In the following, we suggest some additional optimizations.

4.1 Early reporting of test results

In a CCA iteration testing a checkpoint set $\{C_{j,r(j)}|j \in N\}$, the initiator process, a P_i, originally (1) transmits every $S_{j,r(j),j'}$ with $j' \notin \{i,j\}$, (2) tests $\bigwedge_{j\in(N\setminus\{i\})}(R_{i,r(i),j} \leq S_{j,r(j),i})$ and (3) receives the expected responses. After the test, the candidate checkpoint of P_i is a $C_{i,r'(i)}$. If it is different from $C_{i,r(i)}$, it was useless to transmit the counters $S_{i,r(i),j'}$. It is, hence, more appropriate that the test comes before the transmissions, so that P_i can instead transmit the counters $S_{i,r'(i),j'}$. With the *early reporting of test results*, the checkpoint which P_i considers in the current iteration is virtually $C_{i,r'(i)}$, which might spare a subsequent iteration.

Example 3. *With early reporting of test results, the scenario fragment in Example 2 simplifies to the following (see also Fig. 1):*

(8) After P_1 receives the responses, it observes $R_{1,2,2} = 4 > S_{2,2,1} = 3$ and consequently starts considering $C_{1,1}$. It then sends a CM carrying $S_{1,1,2}$ and $S_{3,2,2}$ to P_2 and a CM carrying $S_{1,1,3}$ and $S_{2,2,3}$ to P_3.

(9) The same as (11) in Example 2.

(10) After receiving the two indications, P_1, observing that every flag received was 0, correctly concludes that a GCSLC has been found, tells P_2 and P_3 to restart from $C_{2,2}$ and $C_{3,2}$, respectively, and finally restarts from $C_{1,1}$.

4.2 Immediate counter reporting

Originally, the CMs inviting processes to another instance of the CCA do not comprise the relevant transmission counters of the initiator. We think that they should, because such *immediate counter reporting* might spare an iteration.

Example 4. *Remember that in Example 3, the first seven steps are as in Example 1. If the process which fails in the fifth step is changed to P_2, the subsequent scenario fragment changes to the following (see also Fig. 1):*

(6a) After P_2 recovers, it starts considering its most recent checkpoint $C_{2,2}$ and invites P_1 and P_3 to a new instance of the CCA.

(7a) In response, P_1 starts considering its most recent checkpoint $C_{1,2}$ and sends to P_2 a CM carrying $S_{1,2,2}$ and $S_{1,2,3}$, while P_3 starts considering its most recent checkpoint $C_{3,2}$ and sends to P_2 a CM carrying $S_{3,2,1}$ and $S_{3,2,2}$.

(8a) After P_2 receives the responses, it observes $R_{2,2,1} = 0 \leq S_{1,2,2} = 0$ and $R_{2,2,3} = 0 \leq S_{3,2,2} = 0$ and therefore continues considering $C_{2,2}$. It then sends a CM carrying $S_{2,2,1}$ and $S_{3,2,1}$ to P_1 and a CM carrying $S_{1,2,3}$ and $S_{2,2,3}$ to P_3.

(9a) Upon receiving $S_{2,2,1}$ and $S_{3,2,1}$, P_1 observes $R_{1,2,2} = 4 > S_{2,2,1} = 3$ and consequently starts considering $C_{1,1}$, indicating that to P_2 by a CM carrying flag 1 and $S_{1,1,2}$ and $S_{1,1,3}$. On the other hand, P_3 upon receiving $S_{1,2,3}$ and $S_{2,2,3}$ observes $R_{3,2,1} = 0 \leq S_{1,2,3} = 0$ and $R_{3,2,2} = 0 \leq S_{2,2,3} = 0$ and therefore continues considering $C_{3,2}$, indicating that to P_2 by a CM carrying flag 0 and $S_{3,2,1}$ and $S_{3,2,2}$.

(10a) After receiving the two indications, P_2, observing that a non-zero flag has been received, observes $R_{2,2,1} = 0 \leq S_{1,1,2} = 0$ and $R_{2,2,3} = 0 \leq S_{3,2,2} = 0$, consequently deciding to continue considering $C_{2,2}$, and then sends a CM carrying $S_{2,2,1}$ and $S_{3,2,1}$ to P_1 and a CM carrying $S_{1,1,3}$ and $S_{2,2,3}$ to P_3.

(11a) Upon receiving $S_{2,2,1}$ and $S_{3,2,1}$, P_1 observes $R_{1,1,2} = 0 \leq S_{2,2,1} = 3$ and $R_{1,1,3} = 0 \leq S_{3,2,1} = 7$ and therefore continues considering $C_{1,1}$, indicating that to P_2 by a CM carrying flag 0 and $S_{1,1,2}$ and $S_{1,1,3}$. Likewise, P_3 upon receiving $S_{1,1,3}$ and $S_{2,2,3}$ observes $R_{3,2,1} = 0 \leq S_{1,1,3} = 0$ and $R_{3,2,2} = 0 \leq S_{2,2,3} = 0$ and therefore continues considering $C_{3,2}$, indicating that to P_2 by a CM carrying flag 0 and $S_{3,2,1}$ and $S_{3,2,2}$.

(12a) After receiving the two indications, P_2, observing that every flag received was 0, correctly concludes that a GCSLC has been found, tells P_1 and P_3 to restart from $C_{1,1}$ and $C_{3,2}$, respectively, and finally restarts from $C_{2,2}$.

If immediate counter reporting is introduced, the scenario fragment simplifies to the following:

(6b) After P_2 recovers, it starts considering $C_{2,2}$, sending to P_1 an invitation carrying $S_{2,2,1}$ and to P_3 an invitation carrying $S_{2,2,3}$.

(7b) In response, P_1, observing that $R_{1,2,2} = 4 > S_{2,2,1} = 3$, starts considering $C_{1,1}$ and sends to P_2 a CM carrying $S_{1,1,2}$ and $S_{1,1,3}$, while P_3, observing that $R_{3,2,2} = 0 \leq S_{2,2,3} = 0$, starts considering its most recent checkpoint $C_{3,2}$ and sends to P_2 a CM carrying $S_{3,2,1}$ and $S_{3,2,2}$.

(8b) After P_2 receives the responses, it observes $R_{2,2,1} = 0 \leq S_{1,1,2} = 0$ and $R_{2,2,3} = 0 \leq S_{3,2,2} = 0$ and therefore continues considering $C_{2,2}$. It then sends a CM carrying $S_{2,2,1}$ and $S_{3,2,1}$ to P_1 and a CM carrying $S_{1,1,3}$ and $S_{2,2,3}$ to P_3.

(9b) The same as (11a).

(10b) The same as (12a).

4.3 Update reporting

Like [1], we assume that the initiator process maintains an $(n \times n)$-array variable V in which every component $V_{j,j'}$ with $j \neq j'$ is during GCSLC construction regularly updated to the value of the $S_{j,r(j),j'}$ belonging to the checkpoint $C_{j,r(j)}$ currently considered by P_j. Every process P_j repeatedly contributes values for the j^{th} row and checks the values in the j^{th} column of V. The search for a GCSLC terminates when V stabilizes.

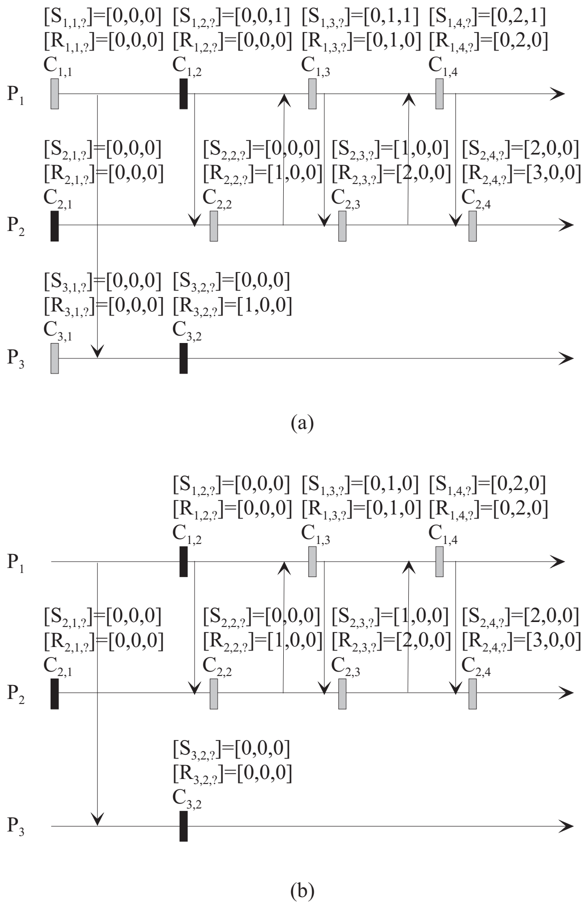
Suppose that a P_j and a $P_{j'}$ are currently considering checkpoints $C_{j,r(j)}$ and $C_{j',r(j')}$, respectively, and that $R_{j,r(j),j'} \leq S_{j',r(j'),j}$, as required. Then suppose that P_j and $P_{j'}$ at some later point move their attention towards some older checkpoints, a $C_{j,r'(j)}$ and a $C_{j',r'(j')}$, respectively. $R_{j,r'(j),j'} \leq R_{j,r(j),j'}$ implies that the problematic $R_{j,r'(j),j'} > S_{j',r'(j'),j}$ is possible only if $S_{j',r'(j'),j} < S_{j',r(j'),j}$, i.e., if $V_{j',j}$ has changed, for $S_{j',r'(j'),j} > S_{j',r(j'),j}$ is impossible. It, hence, suffices that *updates* to $V_{j',j}$ are reported by $P_{j'}$ to the initiator process, a P_i, and received by P_j from P_i. A CM sent to P_i is then a *zero-flag CM* exactly if it carries *no transmission counters*. In [1], a CM carrying information on V unnecessarily always carries an entire row or column, respectively, and flags are sent explicitly.

Example 5. *If only updates to V are reported, the scenario fragment (8b)-(10b) in Example 4 simplifies to the following (see also Fig. 1):*

(8b) After P_2 receives the responses, it observes $R_{2,2,1} = 0 \leq S_{1,1,2} = 0$ and $R_{2,2,3} = 0 \leq S_{3,2,2} = 0$ and therefore continues considering $C_{2,2}$. It then sends a CM carrying $S_{3,2,1}$ to P_1 and a CM carrying $S_{1,1,3}$ to P_3.

(9b) Upon receiving $S_{3,2,1}$, P_1 observes $R_{1,1,3} = 0 \leq S_{3,2,1} = 7$ and therefore continues considering $C_{1,1}$, indicating that to P_2 by a CM carrying no transmission counters. Likewise, P_3 upon receiving $S_{1,1,3}$ observes $R_{3,2,1} = 0 \leq S_{1,1,3} = 0$ and therefore continues considering $C_{3,2}$, indicating that to P_2 by a CM carrying no transmission counters.

(10b) After receiving the two indications, P_2, observing that no transmission counters have been received, correctly concludes that a GCSLC has been found, tells P_1 and P_3 to restart from $C_{1,1}$ and $C_{3,2}$, respectively, and finally restarts from $C_{2,2}$.


$[S_{1,1,?}]=[0,0,0]$ $[S_{1,2,?}]=[0,0,1]$ $[S_{1,3,?}]=[0,1,1]$ $[S_{1,4,?}]=[0,2,1]$
$[R_{1,1,?}]=[0,0,0]$ $[R_{1,2,?}]=[0,0,0]$ $[R_{1,3,?}]=[0,1,0]$ $[R_{1,4,?}]=[0,2,0]$
$C_{1,1}$ $C_{1,2}$ $C_{1,3}$ $C_{1,4}$
P_1
$[S_{2,1,?}]=[0,0,0]$ $[S_{2,2,?}]=[0,0,0]$ $[S_{2,3,?}]=[1,0,0]$ $[S_{2,4,?}]=[2,0,0]$
$[R_{2,1,?}]=[0,0,0]$ $[R_{2,2,?}]=[1,0,0]$ $[R_{2,3,?}]=[2,0,0]$ $[R_{2,4,?}]=[3,0,0]$
$C_{2,1}$ $C_{2,2}$ $C_{2,3}$ $C_{2,4}$
P_2
$[S_{3,1,?}]=[0,0,0]$ $[S_{3,2,?}]=[0,0,0]$
$[R_{3,1,?}]=[0,0,0]$ $[R_{3,2,?}]=[1,0,0]$
$C_{3,1}$ $C_{3,2}$
P_3
$[S_{1,2,?}]=[0,0,0]$ $[S_{1,3,?}]=[0,1,0]$ $[S_{1,4,?}]=[0,2,0]$
$[R_{1,2,?}]=[0,0,0]$ $[R_{1,3,?}]=[0,1,0]$ $[R_{1,4,?}]=[0,2,0]$
$C_{1,2}$ $C_{1,3}$ $C_{1,4}$
P_1
$[S_{2,1,?}]=[0,0,0]$ $[S_{2,2,?}]=[0,0,0]$ $[S_{2,3,?}]=[1,0,0]$ $[S_{2,4,?}]=[2,0,0]$
$[R_{2,1,?}]=[0,0,0]$ $[R_{2,2,?}]=[1,0,0]$ $[R_{2,3,?}]=[2,0,0]$ $[R_{2,4,?}]=[3,0,0]$
$C_{2,1}$ $C_{2,2}$ $C_{2,3}$ $C_{2,4}$
P_2
$[S_{3,2,?}]=[0,0,0]$
$[R_{3,2,?}]=[0,0,0]$
$C_{3,2}$
P_3

(a)

(b)

Figure 2: The situation considered in Example 6 (a) before and (b) after the recovery line is advanced to the black checkpoints, which represent the maximum GCSLC.

4.4 Selective polling

If a CM which the initiator process, a P_i, is originally supposed to send to a process P_j is the j^{th} column of V and P_i discovers that P_j already has a faithful copy of the column, *the CM may be omitted*, for even if received, P_j would continue considering the same checkpoint and produce no update for the j^{th} row of V. Moreover, if P_j does not receive the CM, it will not attempt to send a response, meaning that it will *skip an entire iteration* (see Example 6, step 13). Besides, if P_i discovers that all the other processes may skip the current iteration, it may *immediately indicate CCA termination* (see Example 6, step 15).

Example 6. *With the above optimizations, the following scenario is possible in a system consisting of processes P_1 to P_3 (see also Fig. 2):*

(1) Upon the start of the system, every process P_j takes a checkpoint $C_{j,1}$, with every $S_{j,1,j'}$ and $R_{j,1,j'}$ zero.

(2) P_1 sends an AM to P_3 and takes a checkpoint $C_{1,2}$, with $S_{1,2,3} = 1$ and $S_{1,2,2} = R_{1,2,2} = R_{1,2,3} = 0$.

(3) P_3 receives the AM and takes a checkpoint $C_{3,2}$, with $R_{3,2,1} = 1$ and $S_{3,2,1} = S_{3,2,2} = R_{3,2,2} = 0$.

(4) P_1 sends an AM to P_2. P_2 receives the AM and takes a checkpoint $C_{2,2}$, with $R_{2,2,1} = 1$ and $S_{2,2,1} = S_{2,2,3} = R_{2,2,3} = 0$.

(5) P_2 sends an AM to P_1. P_1 receives the AM and takes a checkpoint $C_{1,3}$, with $S_{1,3,2} = S_{1,3,3} = R_{1,3,2} = 1$ and $R_{1,3,3} = 0$.

(6) P_1 sends an AM to P_2. P_2 receives the AM and takes a checkpoint $C_{2,3}$, with $S_{2,3,1} = 1$, $R_{2,3,1} = 2$ and $S_{2,3,3} = R_{2,3,3} = 0$.

(7) P_2 sends an AM to P_1. P_1 receives the AM and takes a checkpoint $C_{1,4}$, with $S_{1,4,2} = R_{1,4,2} = 2$, $S_{1,4,3} = 1$ and $R_{1,4,3} = 0$.

(8) P_1 sends an AM to P_2. P_2 receives the AM and takes a checkpoint $C_{2,4}$, with $S_{2,4,1} = 2$, $R_{2,4,1} = 3$ and $S_{2,4,3} = R_{2,4,3} = 0$.

(9) P_2 decides to initiate the CCA just for advancing the recovery line. So it starts considering $C_{2,4}$, sending to P_1 an invitation carrying $S_{2,4,1}$ and to P_3 an invitation carrying $S_{2,4,3}$.

(10) In response, P_1, observing that $R_{1,4,2} = 2 \leq S_{2,4,1} = 2$, starts considering $C_{1,4}$ and sends to P_2 a CM carrying $S_{1,4,2}$ and $S_{1,4,3}$, while P_3, observing that $R_{3,2,2} = 0 \leq S_{2,4,3} = 0$, starts considering $C_{3,2}$ and sends to P_2 a CM carrying $S_{3,2,1}$ and $S_{3,2,2}$.

(11) After P_2 receives the responses, it observes $R_{2,4,1} = 3 > S_{1,4,2} = 2$ and therefore starts considering $C_{2,3}$. It then sends a CM carrying $S_{2,3,1}$ and $S_{3,2,1}$ to P_1 and a CM carrying $S_{1,4,3}$ to P_3.

(12) Upon receiving $S_{2,3,1}$ and $S_{3,2,1}$, P_1, observing that $R_{1,4,2} = 2 > S_{2,3,1} = 1$, starts considering $C_{1,3}$, indicating that to P_2 by a CM carrying $S_{1,3,2}$. On the other hand, P_3 upon receiving $S_{1,4,3}$ continues considering $C_{3,2}$, indicating that to P_2 by a CM carrying no transmission counters.

(13) After receiving the two indications, P_2, observing that $R_{2,3,1} = 2 > S_{1,3,2} = 1$, starts considering $C_{2,2}$, sending to P_1 a CM carrying $S_{2,2,1}$. P_3 is not involved in the iteration, *because $S_{1,3,3} = S_{1,4,3}$ and $S_{2,2,3} = S_{2,3,3}$.*

(14) Upon receiving $S_{2,2,1}$, P_1, observing that $R_{1,3,2} = 1 > S_{2,2,1} = 0$, starts considering $C_{1,2}$, indicating that to P_2 by a CM carrying $S_{1,2,2}$.

(15) After receiving the indication, P_2, observing that $R_{2,2,1} = 1 > S_{1,2,2} = 0$, starts considering $C_{2,1}$. It then, because $S_{1,2,3} = S_{1,3,3}$ and $S_{2,1,1} = S_{2,2,1}$ and $S_{2,1,3} = S_{2,2,3}$, immediately *concludes that a GCSLC has been found, indicates that to P_1 and P_3 and finally makes $C_{2,1}$ its recovery-line checkpoint.*

(16) Upon receiving the indication, P_1 makes $C_{1,2}$ its recovery-line checkpoint, meaning that it deletes $C_{1,1}$ and subtracts $S_{1,2,3}$ from $S_{1,2,3}$, $S_{1,3,3}$ and $S_{1,4,3}$. Likewise, P_2 makes $C_{3,2}$ its recovery-line checkpoint, meaning that it deletes $C_{3,1}$ and subtracts $R_{3,2,1}$ from $R_{3,2,1}$. The resulting situation is given in Fig. 2(b).

5 Details of the algorithm

5.1 Introduction

For a CCA instance I, let $P_{In(I)}$ denote its initiator process, Adv_I the predicate telling whether it starts by an attempt to advance the recovery line, and Rst_I the predicate telling whether it terminates by a system restart, i.e., whether it is an instance initiated upon a failure. We assume $Adv_I \vee Rst_I$, so that I has a purpose.

Every CM of a CCA instance I is either an *invitation CM* (ICM) or an *ordinary CM* (OCM). If it is sent by $P_{In(I)}$ and carries no transmission counters, it is also a *termination CM* (TCM).

In Sections 5.2–5.4, we specify a CCA instance, an I, as if it runs in isolation. The given CCA is essentially the CCA of [1] corrected and slightly modified as proposed in Sections 3 and 4. In Section 5.5, we, unlike [1], specify also how processes are assumed to react if they during the execution of a specific CCA instance receive an AM or a CM belonging to another CCA instance.

5.2 Behaviour of the initiator

In a CCA instance I, $P_{In(I)}$, having decided on Adv_I and Rst_I, executes the following sequence of steps, starting at a time $t^1_{I,In(I)}$ and broadcasting a TCM at a time t^2_I:

(1) If $\neg Adv_I$, it broadcasts an ICM carrying just Adv_I (and thereby implicitly also Rst_I) and the current time, i.e. a TCM carrying $t^1_{I,In(I)}$ and t^2_I, and then goes directly to the procedure specified in Section 5.4. Otherwise, it proceeds as follows.

(2) It sets CC, the variable denoting its currently considered checkpoint, to its most recent checkpoint, a $C_{In(I),r}$, and initializes in its $(n \times n)$-array variable V every $V_{In(I),j}$ to $S_{In(I),r,j}$, every $V_{j,j}$ to zero and every other $V_{j,j'}$ to a value so large that no transmission or reception counter can ever acquire it.

(3) To every other process P_j, it sends, and records that by setting a Boolean variable Q_j to true, an ICM carrying Adv_I, Rst_I, $V_{In(I),j}$ and in the case of Rst_I also $t^1_{I,In(I)}$.

(4) It makes a copy V' of V.

(5) From every other process P_j with Q_j true, it receives, and records that by setting Q_j to false, an OCM carrying at least Rst_I and then changes every $V_{j,j'}$ for which the CM carries a value to the value received.

(6) It sets CC to its most recent checkpoint $C_{In(I),r}$ with $R_{In(I),r,j} \leq V_{j,In(I)}$ for every process P_j and sets every $V_{In(I),j}$ to $S_{In(I),r,j}$.

(7) If for every other process P_j, every $V_{j',j}$ is the same as $V'_{j',j}$, it broadcasts an OCM carrying just Rst_I and the current time, i.e. a TCM carrying t^2_I, and then goes directly to the procedure specified in Section 5.4. Otherwise, it proceeds as follows.

(8) To every other process P_j with a $V_{j',j}$ different from $V'_{j',j}$, it sends, and records that by setting Q_j to true, an OCM carrying Rst_I and every such $V_{j',j}$.

(9) It iterates to the step (4).

5.3 Behaviour of the other processes

In a CCA instance I, a process P_j other than $P_{In(I)}$ for every variable $V_{j,j'}$ and $V_{j',j}$ of $P_{In(I)}$ maintains a copy with the same name, executing the following sequence of steps:

(1) It receives, at a time $t^1_{I,j}$, from $P_{In(I)}$ an ICM and sets Adv_I, Rst_I, $t^1_{I,In(I)}$, t^2_I and $V_{In(I),j}$ to the value received, if any. We assume that $(t^1_{I,j} - t^1_{I,In(I)})$ does not exceed a predefined $T^1_{In(I),j}$. An appropriate value for a $T^1_{j',j}$ with $j' \neq j$ would typically be slightly over $T_{j',j}$, while every $T^1_{j,j}$ is by definition zero.

(2) If $\neg Adv_I$, it goes directly to the procedure specified in Section 5.4. Otherwise, it proceeds as follows.

(3) If possible and desired, it takes a fresh checkpoint.

(4) It sets $V_{j,j}$ to zero and every $V_{j',j}$ with $j' \notin \{In(I), j\}$ and $V_{j,j'}$ with $j \neq j'$ to a value so large that no transmission or reception counter can ever acquire it.

(5) It sets its variable CC to its most recent checkpoint $C_{j,r}$ with $R_{j,r,j'} \leq V_{j',j}$ for every process $P_{j'}$, sends to $P_{In(I)}$ an OCM carrying Rst_I and, as a new value for $V_{j,j'}$, every $S_{j,r,j'}$ different from $V_{j,j'}$, and then sets every $V_{j,j'}$ to $S_{j,r,j'}$.

(6) It receives from $P_{In(I)}$ an OCM carrying at least Rst_I and changes every $V_{j',j}$ for which the CM carries a value to the value received. If the CM was a TM, it sets t^2_I to the value received and goes directly to the procedure specified in Section 5.4. Otherwise, it iterates to the step (5).

The time when a process P_j gets involved into a CCA instance I, hence, does not exceed a $t^3_{I,j}$ defined as $min\{(t^1_{I,In(I)} + T^1_{In(I),j}), t^2_I\}$ in the case of Adv_I and as $(t^1_{I,In(I)} + T^1_{In(I),j})$ otherwise.

5.4 Moving the recovery line and/or restarting

Here are the steps of the procedure which in a CCA instance I, every process P_j, with its CC in the case of Adv_I a $C_{j,r}$, executes at the end of the CCA, terminating at a time $t^4_{I,j}$:

(1) If Adv_I, it deletes from its storage every checkpoint older than $C_{j,r}$.

(2) If Rst_I, it deletes from its storage every checkpoint except the oldest one.

(3) For every preserved checkpoint $C_{j,r'}$, it decreases every $S_{j,r',j'}$ by $S_{j,r,j'}$ and every $R_{j,r',j'}$ by $R_{j,r,j'}$.

(4) If Rst_I, it restarts from its oldest preserved checkpoint.

We assume that $(t^4_{I,j} - t^2_I)$ does not exceed a predefined $T^2_{In(I),j}$. An appropriate value for a $T^2_{j',j}$ would typically be slightly over $T_{j',j}$.

5.5 Handling of unexpected messages

If a process waiting for an OCM of a CCA instance I instead receives an AM, it freely decides whether to process it concurrently to or after I.

If a process P_j waiting for an OCM of a CCA instance I instead receives a CM of another CCA instance, an I', it reacts as follows: An ICM with $Rst_{I'}$ false or an OCM with $Rst_{I'} \neq Rst_I$ is just discarded. Upon an ICM with $Rst_{I'}$ true, P_j abandons I and starts executing I' instead. An OCM with $Rst_{I'} = Rst_I$ is erroneously recognized as an OCM of I, so such situation must be prevented (see Section 6.3).

If a process P_j currently involved in no CCA instance receives an OCM or an ICM for which it suspects that it belongs to an obsolete CCA instance, it just discards it. An ICM of a CCA instance I', received at a time t from a process $P_{j'}$, is recognized as obsolete if $\neg Rst_{I'}$ and P_j has participated in a CCA instance I with Rst_I true and $(t^3_{I,j'} + T_{j',j}) \geq t$. This is because the estimated worst-case scenario for I overriding I' is that $P_{j'}$ sends the ICM just before it gets involved into I at the time $t^3_{I,j'}$ and then the ICM reaches P_j with the delay $T_{j',j}$.

6 Optimizing algorithm activation

6.1 Introduction

[1] suggests that in the absence of failures, the CCA is initiated periodically, with the period very long, so that one can assume that when a new CCA instance starts, the previous one, if any, has long been completed. More precisely, [1] specifies that the period is much longer than the checkpointing period of any individual process. [1] also suggests that processes initiate the CCA in turns, so that they never attempt to initiate it concurrently. To implement the policy, [1] has all system processes organized in a *virtual ring* along which the right to initiate the CCA is passed as a *token*, merely by the passing of time. Every reception of the token is interpreted as an obligation to initiate the CCA immediately.

In [1], the frequency of token passing is exactly the desired frequency of running the CCA in the absence of failures. We find this scheme too rigid. On the one hand, forcing processes to take a checkpoint or initiate a recovery-line advancement periodically is seldom optimal. Processes should rather wait for a substantial reason. On the other hand, the initiator token should circulate as fast as possible, so that any process wanting it receives it quickly. Therefore, *we drop the periodicity assumption for checkpointing and recovery-line advancing, and keep the virtual ring just as a means of concurrency control, giving it a more appropriate timing*.

6.2 Reasons for checkpointing and recovery-line advancing

For taking a checkpoint, a typical reason would be that the process has since its last checkpoint accomplished a lot of work or exchanged a lot of AMs.

For advancing the recovery line, we see three reasons:

- A process is running out of stable storage and therefore wants that some of the stored checkpoints become obsolete (remember Section 5.4, step 1).
- A reception or a transmission counter of a process is approaching overflow and the process therefore wants more events to become obsolete for counting (remember Section 5.4, step 3).
- A process wants to reduce the discrepancy between its current state and its recovery-line checkpoint, so that, if a failure occurs, its roll-back in the worst case would not be so drastic.

How often such a reason occurs, depends not only on static parameters, such as the size of the local stable storages and the speed of the processes and the channels, but also on the demands of the executed application, which tend to vary with time.

6.3 Prevention of concurrent instantiations

At any moment, the initiator token is virtually either in transit or resides with one of the processes. If a process P_j possesses the token at a time t, the next time when another process $P_{j'}$ possesses it should ideally be soon, but not until $P_{j'}$ has had a chance to detect whether P_j has initiated the CCA at time t, i.e., not until after $(t + T^1_{j,j'})$. The delay is necessary because when initiating a CCA instance I with $\neg Rst_I$, the process $P_{In(I)}$ must be aware of the most recently initiated previous CCA instance, an I', if any, so that it can, by suitably delaying I, secure that the following is satisfied at the time $t^1_{I,In(I)}$:

- $t^1_{I,In(I)} > t^4_{I',In(I)}$ and $t^1_{I,In(I)} > t^2_{I'} + T^2_{In(I'),j}$ for every other process P_j, so that every process has already terminated execution of I'.
- If $Rst_{I'}$, $t^1_{I,In(I)} > t^3_{I',j} + T_{j,j'}$ for every two processes P_j and $P_{j'}$, so that (1) the CMs which I' made obsolete, if any, have already been received and (2) the ICM of I will not be discarded upon reception (remember Section 5.5).

It might happen that while a process is waiting for an opportunity to initiate a CCA instance I with $\neg Rst_I$, another process initiates a CCA instance I' with $Adv_{I'}$. Upon detecting that, $P_{In(I)}$ should drop its pursuit, for the task of advancing the recovery line is already being taken care of.

7 Organization of the storage

When a process gets involved into a CCA instance, it has to access information on its previous checkpoints. [1] suggests that a process maintains this information primarily in its working memory, so that any access to it can be quick. However, a process having just recovered from a failure can safely rely only on the copy of the information which it has made in its stable storage. How efficiently individual parts of the copy can be accessed, strongly depends on the organization of the storage, which, hence, deserves some discussion.

Obviously, each checkpoint $C_{j,r}$ requires that process P_j stores every $R_{j,r,j'}$ with $j \neq j'$ individually, and not just $\sum_{j' \in (N \setminus \{j\})} R_{j,r,j'}$, as in [1]. For each $C_{j,r}$, P_j would, hence, store vectors $\vec{S}_{j,r}$ and $\vec{R}_{j,r}$ of size $(n-1)$.

If we imitated the policy of [1], every process P_j would for every checkpoint $C_{j,r}$ store in the stable storage the list of all $\vec{R}_{j,r'}$ (originally of all $\sum_{j' \in (N \setminus \{j\})} R_{j,r',j'}$) belonging to a $C_{j,r'}$ not later than $C_{j,r}$, so that, when deciding how far back to move from $C_{j,r}$, P_j could fetch all the necessary information in a single access to the storage. However, this would mean that each element of such a list is stored several times, first in the list of the checkpoint to which it belongs and then in the list of every subsequent checkpoint. With list elements vectors of size $(n-1)$, i.e. not of size 1 as expected in [1], this would be unacceptable for large n.

We think that P_j should maintain a single list, for every $C_{j,r}$ comprising a triplet consisting of $\vec{R}_{j,r}$, $\vec{S}_{j,r}$ and a pointer to all the other details which P_j might need if it actually restarts from $C_{j,r}$. As n is fixed, so can be the size of the triplets, meaning that such a list can be easily maintained and accessed as virtually a single block of consecutive storage locations.

Alternatively, P_j could maintain three lists, one for each kind of the items in the triplets. With this approach, P_j could fetch the stable data more selectively. With all the lists consisting of fixed-sized elements, finding their corresponding elements would still be trivial.

Let us also note that fetching the entire R list in a single step might not always be a good idea, as this might be really a lot of data. In other words, when optimizing communication between processes and their stable storage, one is usually forced to make compromises between the number and the size of the messages, where the optimal strategy depends on the specifics of the system and the current circumstances.

It might happen that a CCA instance I does not advance the recovery line in spite of Adv_I true. In such a case, a process wanting to take fresh checkpoints in spite of running out of stable storage may start deleting its earlier checkpoints, with the only constraint that it must never delete its recovery-line checkpoint or its current candidate for a new recovery-line checkpoint. As the starting checkpoint of any process is stored implicitly, processes can survive even without a stable storage for checkpoints [2], because further checkpoints only increase the probability that in the case of a restart, the required roll-back will not be too drastic.

8 Discussion and conclusions

8.1 Contributions of the paper

We have proposed the following improvements to the CCA of [1], its activation and its storage management:

- A logical flaw has been identified and corrected.
- Instead of exchanging entire rows or columns of the array V, processes now exchange only their updates.
- Whenever possible, processes now skip individual iterations of the algorithm.
- It can no longer happen that the algorithm executes a superfluous iteration.
- Obsolete CMs are now properly recognized and ignored.
- By recognizing the policy of recovery-line advancement and the prevention of concurrent instantiations as two separate issues, we have been able to make the former more flexible and the latter, through faster token circulation, less restrictive. The assumption that processes take their checkpoints periodically has also become unnecessary.
- The organization of the stable storage no longer includes multiple copies of reception counters.
- Process are allowed to replace their old checkpoints with fresh ones.

8.2 Correctness and performance of the algorithm

The proposed CCA is essentially that of [1], except that we properly increased the rigour of checkpoint comparison, in every CM added the missing and removed the redundant information, and removed the CMs which consequently lost every purpose. Hence, if the original CCA is correct up to the rigour of checkpoint comparison, our CCA is also correct.

In [1], it is demonstrated that, for the adopted system topology, the there proposed CCA is an improvement over the CCA of [3], in that the number of checkpoint comparisons grows, linearly, much slower with the average number of checkpoints per process, and linearly instead of quadratically with the number of processes. It is also demonstrated that the proposed CCA is an improvement over the CCA of [2], in that the number of the exchanged CMs grows linearly instead of quadratically with the number of processes.

If the checkpoint comparisons in the CCA of [1] are made sufficiently rigorous, the algorithm preserves all the above advantages. With the proposed additional optimizations of process communication, the number of checkpoint comparisons and the number of the exchanged CMs are sometimes even lower, because the redundant iterations are skipped and because the redundant requests for information on transmission counters are removed. Besides, the remaining CMs are sometimes shorter, because processes exchange just updates to V.

8.3 Concluding remarks

For a distributed algorithm, it is equally important to decide on the task which the process should perform in cooperation, on the protocol securing the necessary coordination and on the management of concurrent instances of the algorithm. For the CCA of [1], we proposed a correction of the task and several improvements to the protocol and the concurrency management. For further work, we propose quantitative assessment of the expected benefits of the additional optimizations.

References

[1] Gupta B., Rahimi S., Yang Y. (2007) A novel roll-back mechanism for performance enhancement of asynchronous checkpointing and recovery, *Informatica*, 31(1), pp. 1–13.

[2] Juang T., Venkatesan, S. (1991) Crash recovery with little overhead, *Proceedings of the 11th International Conference on Distributed Computing Systems*, IEEE, Arlington, Texas, pp. 454–461.

[3] Ohara M., Arai M., Fukumoto S., Iwasaki K. (2004) Finding a recovery line in uncoordinated checkpointing, *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops*, IEEE, Hachioji, Tokyo, Japan, pp. 628–633.