Checkpointing using Local Knowledge about Recovery Lines

Islene Calciolari Garcia
Luiz Eduardo Buzato

Relatório Técnico IC–99-22

Novembro de 1999

Checkpointing using Local Knowledge about Recovery Lines

Islene C. Garcia Luiz E. Buzato
Universidade Estadual de Campinas
Caixa Postal 6176
13083-970 Campinas, SP, Brasil
Tel: +55 19 788 5876
Fax: +55 19 788 5847
{islene,buzato}@dcc.unicamp.br

Abstract

A recovery line is the most recent consistent global checkpoint from which a distributed computation can be restarted after a failure. This paper presents a new quasi-synchronous checkpointing protocol where processes propagate their local knowledge about the recovery line, named PRL.

Protocols that enforce rollback-dependency trackability (RDT) are domino-effect free and are usually based on vector clocks. Protocols that avoid the domino effect without enforcing RDT are usually index-based. The protocol proposed by Baldoni, Quaglia, and Ciciani (BQC) was the first non-RDT domino-effect free protocol to use a vector clock. In BQC, each process keeps and propagates a matrix of $n \times n$ of integers, where n is the number of processes in the computation. PRL is also vector-clock based, domino-effect free, and non-RDT, but lowers the complexity of the required control information from $O(n^2)$ to $O(n)$. We present theoretical and simulation results giving evidence that PRL reduces the number of induced checkpoints in comparison with BQC.

Keywords: distributed checkpointing, quasi-synchronous checkpointing protocols, consistent global states, domino effect, fault tolerance.

1 Introduction

A checkpoint is a stable memory record of a process' state. A consistent global checkpoint is a set of checkpoints, one per process, that could have been seen by an idealized observer external to the computation [5]. A recovery line is the most recent consistent global checkpoint from which a distributed computation can be restarted after a failure.

Fault tolerance based on recovery lines depend on two classes of protocols. One of the classes is concerned with efficient protocols for the recording of checkpoints that can be assembled into a progression of recovery lines. The other class searches efficient protocols for the automation of the resetting of the computation back to the recovery line.

Checkpointing protocols are classified into: asynchronous, quasi-synchronous, and synchronous [14]. When processes take checkpoints asynchronously, the distributed computation is prone to the domino effect, that will force all processes to be backed up to their initial

checkpoint in the worst case [17]. In synchronous checkpointing, processes synchronize their checkpointing activity to construct the recovery line [5, 12]. The major advantage of this method is that the computation is domino-effect free. Major disadvantages are message overhead and possible suspension of process execution during checkpointing coordination.

In a quasi-synchronous checkpointing protocol [1, 2, 3, 4, 6, 9, 11, 14, 22], processes take checkpoints asynchronously (basic checkpoints), but they may be induced by the protocol to take additional checkpoints (forced checkpoints) to enforce that the checkpoints taken satisfy a certain property. The design of checkpointing protocols involves the adoption of two levels of abstraction. At the higher level, the designer analyses distributed computations and tries to infer the points where forced checkpoints have to be taken. At the lower level, the designer decides the data structures that must be maintained and propagated by the processes to allow the evaluation of the condition that induces forced checkpoints.

The property of rollback-dependency trackability (RDT) [18, 22] guarantees that all dependencies among checkpoints are causal dependencies; RDT protocols generally use vector clocks to track dependencies among checkpoints [1, 11, 22]. A weaker property is to avoid the domino effect by preventing useless checkpoints (checkpoints that cannot be part of any consistent global checkpoint) [16]. Protocols that avoid the domino effect but do not enforce RDT are usually index-based protocols [3, 4, 9]; indexes represent a dependency tracking mechanism similar to Lamport's logical clocks [13].

The protocol proposed by Baldoni, Quaglia, and Ciciani (BQC) [2] was the first protocol to use a vector clock and to avoid the domino effect without enforcing RDT. BQC has been developed considering checkpoint intervals and zigzag paths [16]. In BQC, it is important to know in which checkpoint interval a process sent a message to another process; to keep and propagate this information, each process has a matrix of $n \times n$ of integers, where n is the number of processes in the computation. So, the complexity of the protocol is $O(n^2)$.

To our knowledge, the protocol presented in this work is the second protocol to use a vector clock and to avoid the domino effect without enforcing RDT. Our protocol, named PRL, attains this goal with an $O(n)$ complexity through the propagation of local knowledge about recovery lines. To establish the condition necessary to avoid the domino effect, we have used Z-precedence and progressive views [7]. This approach focuses on the use of causal precedence among checkpoints to analyze distributed computations, in contrast with the approach usually found in the literature, that is based on sequences of messages [2, 14, 16].

The paper is structured as follows. Section 2 describes the computational model adopted. Section 3 introduces progressive views and recovery lines. Section 4 describes our protocol and Section 5 compares it with BQC [2]. Section 6 is a summary of the results presented.

2 Computational model

A distributed computation is composed of n sequential processes $(p_0, \ldots, p_{n-1})$ that communicate only by exchanging messages. Messages cannot be corrupted, but can be delivered out of order or lost. A process is modeled as a sequence of *events* that can be divided into internal events and communication events realized through the sending and the reception of messages. Checkpoints are internal events; each process takes an initial checkpoint (imme-

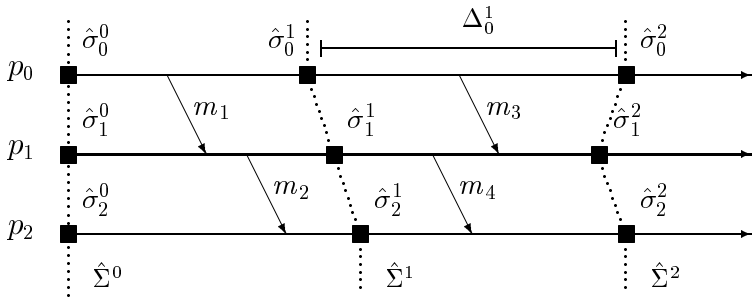


Figure 1: A distributed computation

diately after execution begins) and a final checkpoint (immediately before execution ends). Figure 1 illustrates a distributed computation as a space-time diagram [13] augmented with checkpoints (black squares).

Let $\hat{\sigma}_i^\gamma$ denote the γth checkpoint taken by p_i. Checkpoint $\hat{\sigma}_i^\gamma$ and its immediate successor $\hat{\sigma}_i^{\gamma+1}$ define a checkpoint interval Δ_i^γ. This interval represents the set of events produced by p_i between $\hat{\sigma}_i^\gamma$ and $\hat{\sigma}_i^{\gamma+1}$. Considering Lamport's causal precedence [13], let $\hat{\sigma}_a^\alpha \rightarrow \hat{\sigma}_b^\beta$ indicate that $\hat{\sigma}_a^\alpha$ causally precedes $\hat{\sigma}_b^\beta$. A causal precedence $\hat{\sigma}_a^\alpha \rightarrow \hat{\sigma}_b^\beta$, $a \neq b$, must be due to a sequence of messages $m_1, \ldots, m_k$ from $\hat{\sigma}_a^\alpha$ to $\hat{\sigma}_b^\beta$, e.g., $\hat{\sigma}_0^0 \rightarrow \hat{\sigma}_2^1$ is due to m_1, m_2 (Figure 1).

Definition 2.1 Causal sequence of messages—*A causal precedence $\hat{\sigma}_a^\alpha \rightarrow \hat{\sigma}_b^\beta$ is due to a causal sequence of messages $\mu = (m_1, \ldots, m_k)$ if p_a sends m_1 after $\hat{\sigma}_a^\alpha$, the reception of m_i, $1 \leq i < k$ precedes the sending of m_{i+1} in the same process, and p_b receives m_k before $\hat{\sigma}_b^\beta$.*

Since a consistent global checkpoint could have been seen by an idealized external observer, it must contain only causally unrelated (concurrent) checkpoints [5]. In Figure 1, $\hat{\Sigma}^0$, $\hat{\Sigma}^1$, and $\hat{\Sigma}^2$ are examples of consistent global checkpoints.

Definition 2.2 Consistent Global Checkpoint—*A global checkpoint $\hat{\Sigma} = \{\hat{\sigma}_0^{\iota_0}, \ldots, \hat{\sigma}_{n-1}^{\iota_{n-1}}\}$ is consistent iff*

$$\forall i, j : 0 \leq i, j < n : (\hat{\sigma}_i^{\iota_i} \not\rightarrow \hat{\sigma}_j^{\iota_j})$$

3 Progressive Views and Recovery Lines

A progressive view of a distributed computation is a sequence of consistent global checkpoints such that each global checkpoint in the sequence appears to have happened after the other [7]. For example, the sequence $(\hat{\Sigma}^0, \hat{\Sigma}^1, \hat{\Sigma}^2)$ forms a progressive view of the computation depicted in Figure 1. The restriction imposed on this sequence is that if a checkpoint x causally precedes a checkpoint y, a consistent global checkpoint that includes y cannot precede a consistent global checkpoint that includes x in the sequence.

Definition 3.1 Progressive View—*A progressive view of a distributed computation is a sequence of consistent global checkpoints $(\hat{\Sigma}^0, \hat{\Sigma}^1, \ldots, \hat{\Sigma}^m)$ such that*

$$\forall k : 0 \leq k < m : (\hat{\sigma} \in \hat{\Sigma}^k) \wedge (\hat{\sigma}' \in \hat{\Sigma}^{k+1}) \Rightarrow (\hat{\sigma}' \not\rightarrow \hat{\sigma})$$

The restriction imposed on the sequence of consistent global checkpoints leads to a well-defined order for the observation of checkpoints. A checkpoint x must be *observed before* a checkpoint y in a progressive view if y cannot be part of a consistent global checkpoint with x or with a preceding checkpoint of x in the same process.

Definition 3.2 Observed Before—*A checkpoint* $\hat{\sigma}_a^\alpha$ *must be* observed before *a checkpoint* $\hat{\sigma}_b^\beta$ *iff* $\hat{\sigma}_b^\beta$ *cannot be part of the same consistent global checkpoint with* $\hat{\sigma}_a^\gamma$ *such that* $\gamma \leq \alpha$.

3.1 Z-precedence

If a checkpoint x causally precedes a checkpoint y, x must be *observed before* y. However, concurrent checkpoints may also have a well-defined order in a progressive view; consider checkpoints $\hat{\sigma}_0^1$ and $\hat{\sigma}_2^2$ in Figure 1. Due to m_4, $\hat{\sigma}_1^1$ must be observed before $\hat{\sigma}_2^2$. Thus, $\hat{\sigma}_1^2$ must be observed before or simultaneously to $\hat{\sigma}_2^2$. Due to m_3, $\hat{\sigma}_0^1$ must be observed before $\hat{\sigma}_1^2$. Thus, $\hat{\sigma}_0^2$ must be observed before or simultaneously to $\hat{\sigma}_1^2$. Consequently, $\hat{\sigma}_0^1$ must be observed before $\hat{\sigma}_2^2$. Extending this scenario, the Z-precedence captures exactly whether a checkpoint must be observed before another in a progressive view [7].

Definition 3.3 Z-precedence between checkpoints
Checkpoint $\hat{\sigma}_a^\alpha$ Z-precedes *checkpoint* $\hat{\sigma}_b^\beta$ *(*$\hat{\sigma}_a^\alpha \rightsquigarrow \hat{\sigma}_b^\beta$*) iff*

1. $\hat{\sigma}_a^\alpha \rightarrow \hat{\sigma}_b^\beta$, *or*
2. $\exists \hat{\sigma}_c^\gamma : (\hat{\sigma}_a^\alpha \rightsquigarrow \hat{\sigma}_c^\gamma) \wedge (\hat{\sigma}_c^{\gamma-1} \rightsquigarrow \hat{\sigma}_b^\beta)$

A Z-precedence is a composition of causal precedences. For example, the Z-precedence $\hat{\sigma}_0^1 \rightsquigarrow \hat{\sigma}_2^2$ in Figure 1 can be expressed as $\hat{\sigma}_0^1 \rightarrow \hat{\sigma}_1^2$ and $\hat{\sigma}_1^1 \rightarrow \hat{\sigma}_2^2$. Since it cannot be expressed with a shorter sequence of causal precedences, the *size* of this Z-precedence is 2.

Definition 3.4 Size of a Z-precedence—*A Z-precedence* $\hat{\sigma} \rightsquigarrow \hat{\sigma}'$ *has size* ℓ $(|\hat{\sigma} \rightsquigarrow \hat{\sigma}'| = \ell)$ *if* ℓ *is the minimum number of causal precedences necessary to express it.*

3.2 Recovery Lines

A recovery line is the most recent consistent checkpoint from which a distributed computation can be restarted after a failure; it is also the last consistent global checkpoint in a progressive view. For example, the consistent global checkpoint $\hat{\Sigma}^2$ is the recovery line of the computation depicted in Figure 1.

The progression of the recovery line is equivalent to the construction of the next consistent global checkpoint in a progressive view [7]. In Figure 2 (a) the recovery line cannot advance because checkpoint $\hat{\sigma}_a^\alpha$ must be observed before $\hat{\sigma}_c^\gamma$ and $\hat{\sigma}_b^\beta$. When p_a takes a checkpoint $\hat{\sigma}_a^{\alpha+1}$, as depicted in Figure 2 (b), the recovery line can advance to $\{\hat{\sigma}_a^{\alpha+1}, \hat{\sigma}_c^\gamma, \hat{\sigma}_b^\beta\}$.

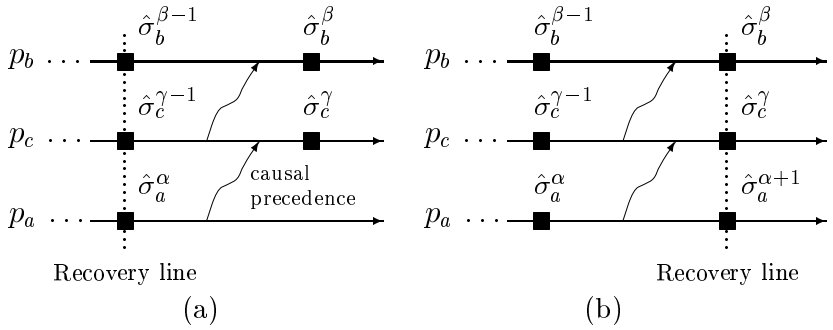


Figure 2: Progression of the recovery line

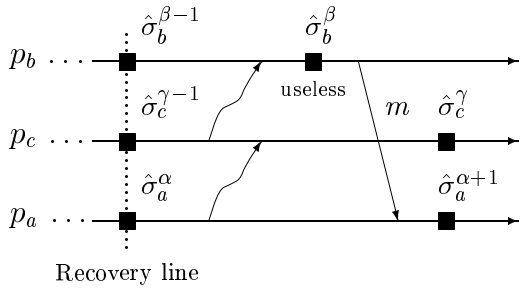


Figure 3: Useless checkpoint and domino effect

3.3 Useless Checkpoints and Domino Effect

If processes take checkpoints independently, the recovery line may not progress, resulting in the undesirable domino effect [17]. Consider the scenario depicted in Figure 3. As explained in the previous Section, $\hat{\sigma}_a^{\alpha}$ must be observed before $\hat{\sigma}_b^{\beta}$. Also, due to m, $\hat{\sigma}_b^{\beta}$ must be observed before $\hat{\sigma}_a^{\alpha+1}$. This scenario corresponds to a Z-cycle: $\hat{\sigma}_b^{\beta}$ Z-precedes itself. Since a checkpoint cannot be observed before itself, this checkpoint is *useless*: it cannot be part of any consistent global checkpoint [7].

Definition 3.5 Z-cycle—*A checkpoint $\hat{\sigma}$ participates in a Z-cycle if, and only if, $\hat{\sigma} \rightsquigarrow \hat{\sigma}$.*

The necessary and sufficient condition under which a checkpoint is useless was first stated by Netzer and Xu [16]. They characterize Z-cycles using zigzag paths formed by sequences of messages connecting checkpoint intervals. Z-precedence is equivalent to zigzag paths, but it represents a higher-level abstraction, because it is based on causal precedences instead of messages [7].

A Z-cycle free checkpoint pattern guarantees that every checkpoint will be part of at least one consistent global checkpoint, but every process must have a final checkpoint that

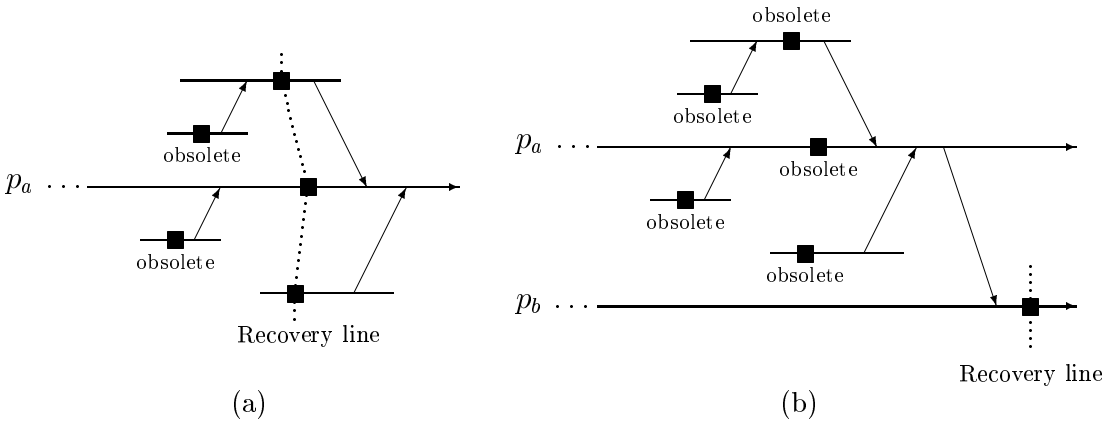


Figure 4: (a) p_a's perspective and (b) p_b's perspective of the recovery line

does not precede any other checkpoint in the pattern [7]. For example, in Figure 2 (a) $\hat{\sigma}_b^\beta$ does not participate in a Z-cycle, but cannot be part of the recovery line. In contrast, in Figure 3 $\hat{\sigma}_b^\beta$ cannot be part of the recovery line despite of the progression of the other processes in the application. Thus, a Z-cycle free checkpointing protocol cannot guarantee that the last checkpoint of a process will be included in the recovery line.

4 Quasi-Synchronous Propagation of Recovery-Lines

The principle of the protocol From the perspective of a process p_a, its most recent checkpoint should belong to the recovery line. Checkpoints that causally precede this checkpoint are *obsolete*, they cannot be part of the recovery line. During a checkpoint interval, p_a may receive knowledge about other checkpoints that belong to the recovery line and also about other obsolete checkpoints (Figure 4 (a)). The local knowledge about the recovery line may greatly differ among processes. Checkpoints that belong to the recovery line as observed by p_a (Figure 4 (a)) are obsolete from the perspective of p_b (Figure 4 (b)). The local information can be quite different from the "real" recovery line as seen by an external observer (Section 3).

Consider process p_a receives a message m informing, for the first time, about the obsolescence of checkpoint $\hat{\sigma}_c^{\gamma-1}$. At the moment m is received, either $\hat{\sigma}_c^{\gamma-1}$ is unknown to p_a (Figure 5 (a)) or it is considered by p_a as non-obsolete (Figure 5 (b)). From the perspective of p_a, a Z-precedence $\hat{\sigma}_a^\alpha \rightsquigarrow \hat{\sigma}_c^\gamma$ could have been established using the message(s) it sent in the interval, possibly resulting in a Z-cycle $\hat{\sigma}_b^\beta \rightsquigarrow \hat{\sigma}_b^\beta$. To avoid this Z-cycle, p_a takes a forced checkpoint $\hat{\sigma}_a^{\alpha+1}$ before delivering m. This cooperation among processes is sufficient to produce a Z-cycle free checkpoint pattern (Theorem 4.1).

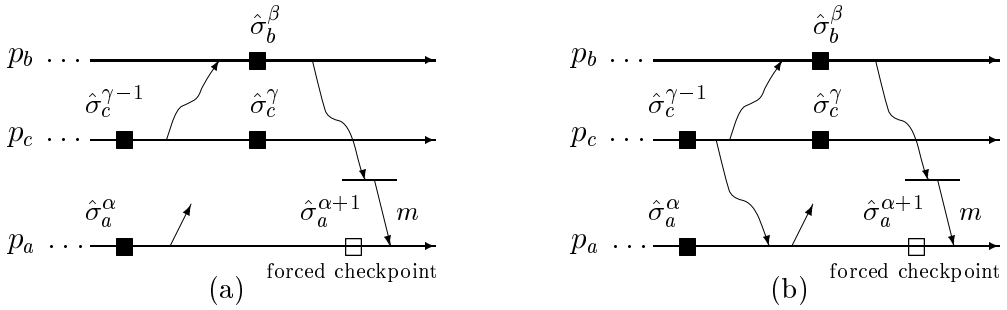


Figure 5: PRL must take a forced checkpoint

Data Structures The protocol is specified in class `PRL` (Class 4.1), using Java[1] [8, 19]. Every process maintains and propagates a vector clock VC in order to characterize casual precedence among checkpoints [15]. When a message is sent, the vector clock of the sender is piggybacked onto it. Before consuming a message, each process takes a component-wise maximum of its vector clock and the received vector clock. When a process takes a checkpoint, it increments its corresponding entry in the vector clock.

The knowledge about the obsoleteness of checkpoints is maintained using a vector of booleans named *obsolete*. When process p_a takes a checkpoint it sets all entries in *obsolete* to true, except its ath entry. When p_a receives a message, it updates *obsolete* considering the vector clock of the received message. Process p_a also maintains a boolean variable *afterSend* that indicates whether or not a message has been sent in the checkpoint interval.

Statement of protocol condition A checkpoint is induced by p_a before delivering a message m if the following condition holds:

$(\mathcal{C})$ $afterSend_a \wedge$
$\exists c : obsolete_m[c] \wedge (VC_a[c] < VC_m[c] \vee (VC_a[c] = VC_m[c] \wedge \neg\, obsolete_a[c]))$

Domino-effect free property Theorem 4.1 proves that this protocol guarantees the usefulness of all checkpoints.

Theorem 4.1 *PRL is a Z-cycle free checkpointing protocol.*

Proof: Consider a Z-cycle $\hat{\sigma}_b^\beta \rightsquigarrow \hat{\sigma}_b^\beta$ and a process p_a that *closes* this Z-cycle in an interval Δ_a^α, i.e., $\hat{\sigma}_b^\beta \rightarrow \hat{\sigma}_a^{\alpha+1}$ and $\hat{\sigma}_a^\alpha \rightsquigarrow \hat{\sigma}_b^\beta$. The proof uses induction on the size ℓ of $\hat{\sigma}_b^\beta \rightsquigarrow \hat{\sigma}_b^\beta$.

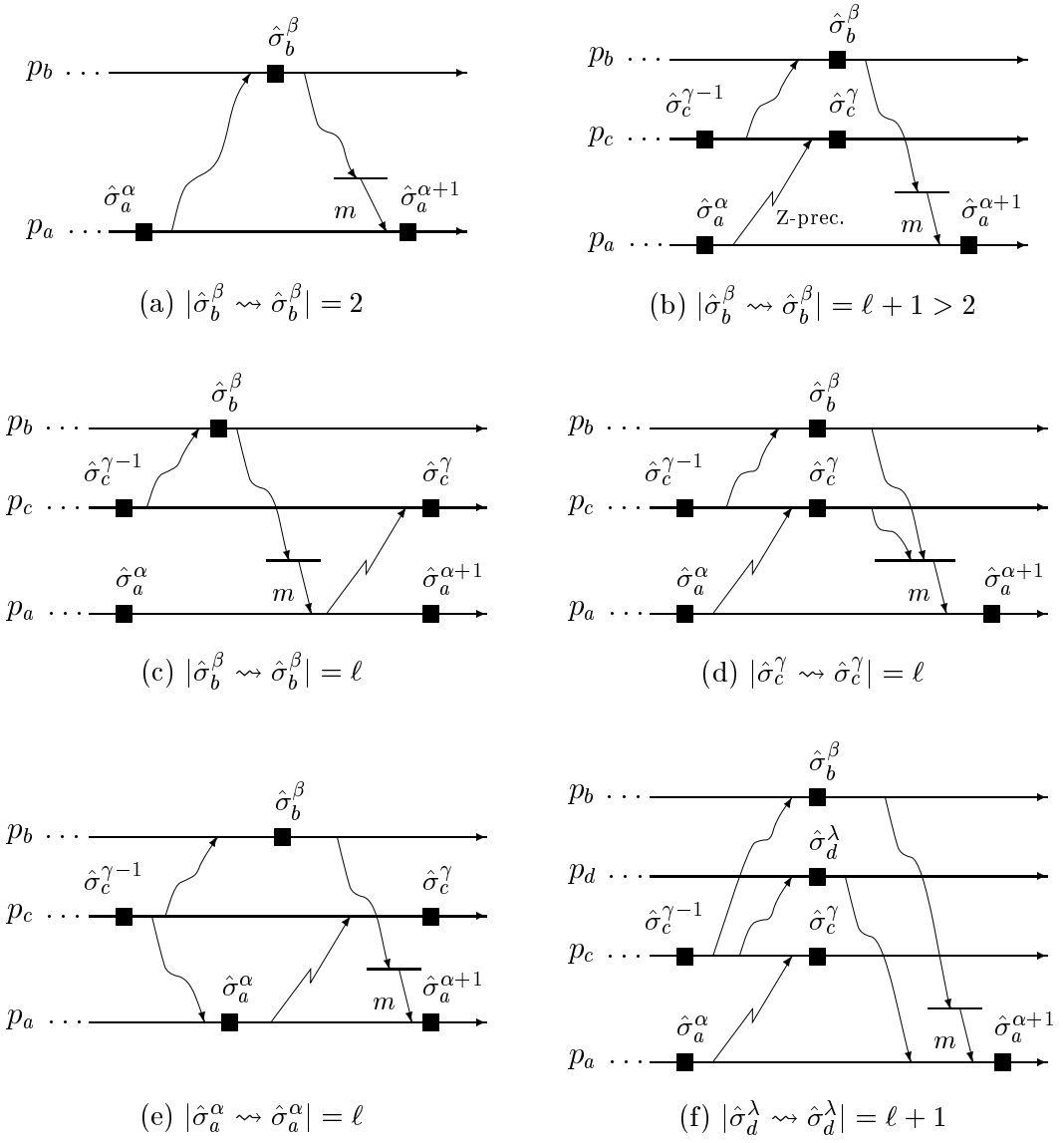
Base: ($\ell = 2$) This Z-cycle can be written as $\hat{\sigma}_b^\beta \rightarrow \hat{\sigma}_a^{\alpha+1}$ and $\hat{\sigma}_a^\alpha \rightarrow \hat{\sigma}_b^\beta$ (Figure 6 (a)). When a message m carrying information about $\hat{\sigma}_b^\beta$ arrives at p_a, we must have this configuration:

- $afterSend_a$—p_a sends the first message related to $\hat{\sigma}_a^\alpha \rightarrow \hat{\sigma}_b^\beta$ before the arrival of m.

[1] We have chosen this language because it is easy to read and has a precise specification. Java is a trademark of Sun Microsystems, Inc.

Class 4.1 PRL.java

```
public class PRL {

    public static final int N = 100;  // Number of processes in the application
    public int pid;  // A unique process identifier in the range 0..N-1
    public int[ ] VC = new int[N];  // Process' vector clock
    public boolean obsolete[ ] = new boolean[N];  // Keeps track of obsolete checkpoints
    public boolean afterSend;  // Indicates whether a message has been sent in the interval

    public void takeCheckpoint() {
        VC[pid]++;
        for (int i=0; i < N; i++) obsolete[i] = i == pid;
        afterSend = false;
        // Save selected state to stable memory
    }

    public PRL(int pid) {  // Constructor
        this.pid = pid;
        for (int i=0; i < N; i++) VC[i] = -1;  // Vector clock initialization
        takeCheckpoint();  // VC[pid] is set to 0, obsolete and afterSend are initialized
    }

    public void finalize() { takeCheckpoint(); }  // Destructor-like method

    public void sendMessage(Message m) {
        m.VC = (int[ ]) VC.clone();  // Copies the whole array
        m.obsolete = (boolean[ ]) obsolete.clone();
        afterSend = true;
        // Send message
    }

    private boolean mustTakeForcedCheckpoint(Message m) {
        if (afterSend)
            for (int i=0; i < N; i++)
                if (m.obsolete[i] && (VC[i] < m.VC[i] || (VC[i] == m.VC[i] && !obsolete[i])))
                    return true;
        return false;
    }

    public void receiveMessage(Message m) {
        if (mustTakeForcedCheckpoint(m)) takeCheckpoint();
        for (int i=0; i < N; i++)  // Component-wise maximum
            if (m.VC[i] > VC[i]) {
                VC[i] = m.VC[i]; obsolete[i] = m.obsolete[i];
            } else if (m.VC[i] == VC[i])
                obsolete[i] = obsolete[i] || m.obsolete[i];
        // Receive message
    }
}
```



(a) $|\hat{\sigma}_b^\beta \rightsquigarrow \hat{\sigma}_b^\beta| = 2$

(b) $|\hat{\sigma}_b^\beta \rightsquigarrow \hat{\sigma}_b^\beta| = \ell + 1 > 2$

(c) $|\hat{\sigma}_b^\beta \rightsquigarrow \hat{\sigma}_b^\beta| = \ell$

(d) $|\hat{\sigma}_c^\gamma \rightsquigarrow \hat{\sigma}_c^\gamma| = \ell$

(e) $|\hat{\sigma}_a^\alpha \rightsquigarrow \hat{\sigma}_a^\alpha| = \ell$

(f) $|\hat{\sigma}_d^\lambda \rightsquigarrow \hat{\sigma}_d^\lambda| = \ell + 1$

Figure 6: Scenarios used in the proof of Theorem 4.1

- $VC_a[a] = \alpha$ and $\neg\, obsolete_a[a]$—Δ_a^α is the current interval of p_a.
- $VC_m[a] = \alpha$ and $obsolete_m[a]$—p_b considers $\hat{\sigma}_a^\alpha$ as an obsolete checkpoint.

According to $\mathcal{C}$, p_a should have taken a forced checkpoint before processing m. Thus, a Z-cycle of size 2 is not allowed by the protocol.

Step: ($\ell \geq 2$) Assume that a Z-cycle of size ℓ is not allowed; we prove that a Z-cycle of size $\ell + 1$ cannot be formed. Consider that $\hat{\sigma}_b^\beta \rightsquigarrow \hat{\sigma}_b^\beta$ can be expressed as $\hat{\sigma}_b^\beta \rightarrow \hat{\sigma}_a^{\alpha+1}$, $\hat{\sigma}_a^\alpha \rightsquigarrow \hat{\sigma}_c^\gamma$ and $\hat{\sigma}_c^{\gamma-1} \rightarrow \hat{\sigma}_b^\beta$, with $|\hat{\sigma}_a^\alpha \rightsquigarrow \hat{\sigma}_c^\gamma| = \ell - 1$ (Figure 6 (b)). Assume, without loss of generality, that $\hat{\sigma}_b^\beta \rightsquigarrow \hat{\sigma}_b^\beta$ is the first Z-cycle of size $\ell + 1$ that p_a closes in the interval Δ_a^α. When a message m carrying information about $\hat{\sigma}_c^{\gamma-1}$ arrives at p_a, we must have the following configuration:

- $afterSend_a$—p_a sends the first message related to $\hat{\sigma}_a^\alpha \rightsquigarrow \hat{\sigma}_c^\gamma$ before the arrival of m. Otherwise, $|\hat{\sigma}_b^\beta \rightsquigarrow \hat{\sigma}_c^\gamma| = |\hat{\sigma}_a^\alpha \rightsquigarrow \hat{\sigma}_c^\gamma| = \ell - 1$ and $|\hat{\sigma}_b^\beta \rightsquigarrow \hat{\sigma}_b^\beta| = \ell$ (Figure 6 (c)).
- $VC_m[c] = \gamma - 1$ and $VC_a[c] \leq \gamma - 1$—The concatenation of $\hat{\sigma}_c^\gamma \rightarrow \hat{\sigma}_a^{\alpha+1}$ and $\hat{\sigma}_a^\alpha \rightsquigarrow \hat{\sigma}_c^\gamma$ is a Z-cycle of size ℓ (Figure 6 (d)).
- $obsolete_m[c]$—p_b considers $\hat{\sigma}_c^{\gamma-1}$ as an obsolete checkpoint.
- $VC_a[c] = \gamma - 1$ and $obsolete_a[c]$—p_a does not take a checkpoint before processing m.

We can also conclude that $\hat{\sigma}_c^{\gamma-1} \not\rightarrow \hat{\sigma}_a^\alpha$, because $\hat{\sigma}_a^\alpha \rightsquigarrow \hat{\sigma}_c^\gamma$ and $\hat{\sigma}_c^{\gamma-1} \rightarrow \hat{\sigma}_a^\alpha$ forms a Z-cycle of size ℓ (Figure 6 (e)). Consequently, the knowledge that $\hat{\sigma}_c^{\gamma-1}$ is an obsolete checkpoint came from another process p_d, and there exists a checkpoint $\hat{\sigma}_d^\lambda$ such that $\hat{\sigma}_c^{\gamma-1} \rightarrow \hat{\sigma}_d^\lambda$ (Figure 6 (f)). Since $\hat{\sigma}_d^\lambda \rightarrow \hat{\sigma}_a^{\alpha+1}$, $\hat{\sigma}_a^\alpha \rightsquigarrow \hat{\sigma}_c^\gamma$, and $\hat{\sigma}_c^{\gamma-1} \rightarrow \hat{\sigma}_d^\lambda$, this contradicts the hypothesis that the Z-cycle involving $\hat{\sigma}_b^\beta$ is the first Z-cycle of size $\ell + 1$ that p_a closes. □

5 A comparison with BQC

Suspect Z-Cycles The protocol proposed by Baldoni, Quaglia, and Ciciani (BQC) is based on the notion of a particular checkpoint and communication pattern called *Suspect Z-Cycle* (SZC) [2]. They proved that the existence of a Z-cycle implies the existence of an SZC, but an SZC is not necessarily part of a Z-cycle.

Definition 5.1 Suspect Z-Cycle (SZC)—*An SZC is a checkpoint and communication pattern SZC($\Delta_a^\alpha, \hat{\sigma}_b^\beta, \Delta_c^{\gamma-1}$) such that*

1. $\exists m : p_c$ *sends* m *in* $\Delta_c^{\gamma-1}$ *and* p_b *receives* m *before* $\hat{\sigma}_b^\beta$.
2. $\exists \mu : \mu$ *is the first causal sequence of messages bringing to* p_a *the knowledge of existence of* $\hat{\sigma}_b^\beta$ *(*μ *is* prime*) and* p_a *receives the last message of* μ *in* Δ_a^α.
3. $\nexists e \in \Delta_c^\gamma : e$ *is an event that causally precedes the delivery of the last message of* μ.

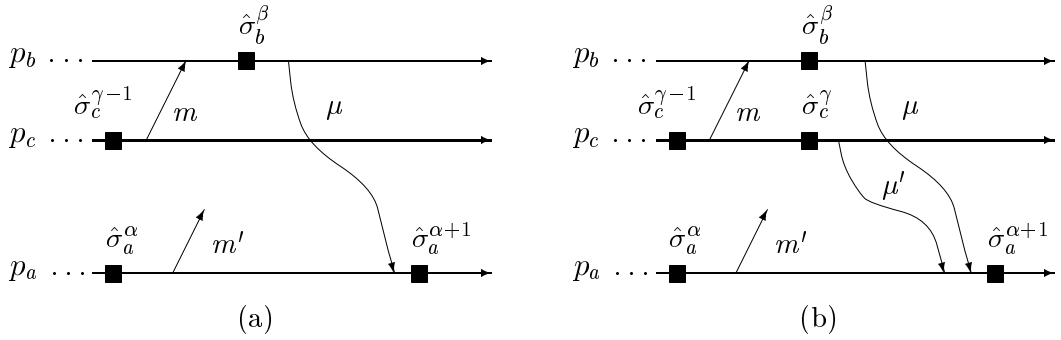


Figure 7: An example (a) and a counter-example (b) of Suspect Z-Cycles

4. $\exists m' : m'$ *was sent in* Δ_a^{α} *before the delivery of the last message of* μ.

Figure 7(a) shows an example of an SZC, while the scenario depicted in Figure 7(b) does not represent an SZC due to the causal sequence of messages μ'.

Analysis of PRL and BQC conditions BQC takes a forced checkpoint upon the detection of an SZC in order to guarantee a ZCF checkpoint pattern. In contrast, PRL induces a checkpoint when a process knows, for the first time, about an obsolete checkpoint. In Figure 7(a), both protocols induce a checkpoint upon the arrival of the last message of μ, because it is the first time p_a knows that $\hat{\sigma}_c^{\gamma-1}$ is obsolete. In Figure 7(b), neither of the protocols induce a checkpoint.

Theorem 5.1 *If PRL takes a forced checkpoint, BQC must also induce a forced checkpoint.*

Proof: Consider that p_a has sent a message m' in its current interval and that it is induced by PRL to take a forced checkpoint before processing m'' (Figure 8). Thus, message m'' must bring to p_a the knowledge of an obsolete checkpoint $\hat{\sigma}_c^{\gamma-1}$ such that p_a had no information about it or p_a considered it as non-obsolete. Also, m'' must bring to p_b the knowledge of a non-obsolete checkpoint $\hat{\sigma}_b^{\beta}$ such that $\hat{\sigma}_c^{\gamma-1} \rightarrow \hat{\sigma}_b^{\beta}$. Message m'' must be the last message of a prime causal sequence of messages from $\hat{\sigma}_b^{\beta}$ to p_a, otherwise p_a would consider $\hat{\sigma}_c^{\gamma-1}$ as obsolete. Let $m_1, \ldots, m_k$ be a causal sequence of messages related to $\hat{\sigma}_c^{\gamma-1} \rightarrow \hat{\sigma}_b^{\beta}$. We prove, using induction on the size of this sequence, that an SZC must be detected by BQC.

Base: $(k = 1)$ There is a message m_1 from $\hat{\sigma}_c^{\gamma}$ to $\hat{\sigma}_b^{\beta}$, m'' is the last message of a prime causal message from $\hat{\sigma}_b^{\beta}$, p_a has no information about $\hat{\sigma}_c^{\gamma}$, and p_a has sent a message m' in the current interval (Figure 8 (a)). This scenario characterizes an SZC $(\Delta_a^{\alpha}, \hat{\sigma}_b^{\beta}, \Delta_c^{\gamma-1})$.

Step: $(k > 1)$ Assume that a sequence of $k-1$ messages from $\hat{\sigma}_c^{\gamma-1} \rightarrow \hat{\sigma}_b^{\beta}$ implies the existence of an SZC. Consider that it exists a sequence of k messages connecting $\hat{\sigma}_c^{\gamma-1}$ to $\hat{\sigma}_b^{\beta}$ and process p_d sent the last message of this sequence, m_k, in its checkpoint interval $\Delta_d^{\lambda-1}$ (Figure 8 (b)). We have the following possibilities concerning the knowledge about checkpoint $\hat{\sigma}_d^{\lambda}$:

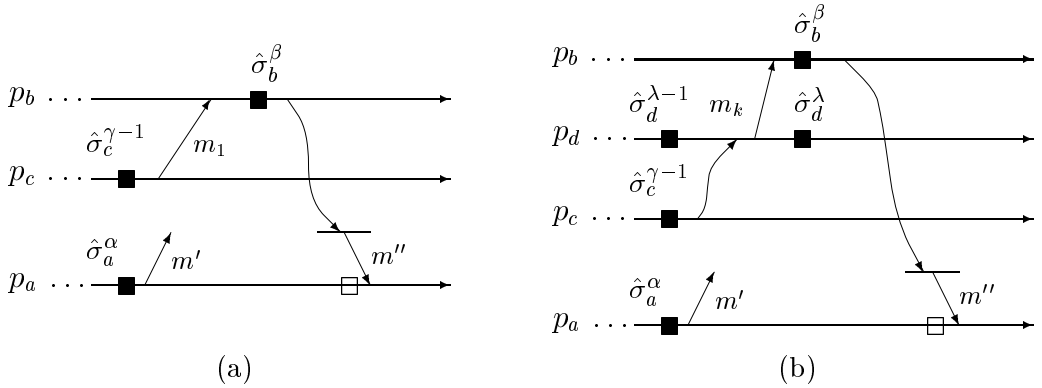


Figure 8: Base and induction of Theorem 5.1

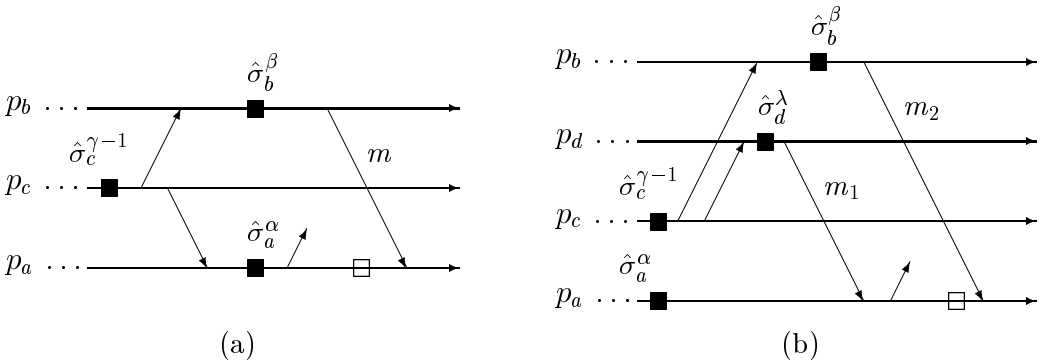


Figure 9: BQC must take an additional forced checkpoint

- Process p_a knows $\hat{\sigma}_d^\lambda$. In this case, since $\hat{\sigma}_c^{\gamma-1} \rightarrow \hat{\sigma}_d^\lambda$, p_a must consider $\hat{\sigma}_c^{\gamma-1}$ as an obsolete checkpoint before the reception of m''.
- Process p_a does not know $\hat{\sigma}_d^\lambda$:
  - Message m'' carries information about $\hat{\sigma}_d^\lambda$. There is a sequence of $k-1$ messages connecting $\hat{\sigma}_c^{\gamma-1} \rightarrow \hat{\sigma}_d^\lambda$. According to the induction hypothesis, BQC would detect an SZC$(\Delta_a^\alpha, \hat{\sigma}_d^\lambda, \Delta_c^{\gamma-1})$.
  - Message m'' does not carry information about $\hat{\sigma}_d^\lambda$. Using the base of the induction, there is an SZC$(\Delta_a^\alpha, \hat{\sigma}_b^\beta, \Delta_d^{\lambda-1})$. □

PRL uses a *stronger* condition to induce checkpoints. In Figure 9 (a), upon the arrival of m, p_a detects an SZC$(\Delta_a^\alpha, \hat{\sigma}_b^\beta, \Delta_c^{\gamma-1})$ and it considers $\hat{\sigma}_c^{\gamma-1}$ as obsolete due to $\hat{\sigma}_a^\alpha$. In Figure 9 (b), upon the arrival of m_2, p_a detects SZC$(\Delta_a^\alpha, \hat{\sigma}_b^\beta, \Delta_c^{\gamma-1})$ and it considers $\hat{\sigma}_c^{\gamma-1}$ as obsolete due to m_1. In both scenarios, BQC induces a forced checkpoint, but PRL does not.

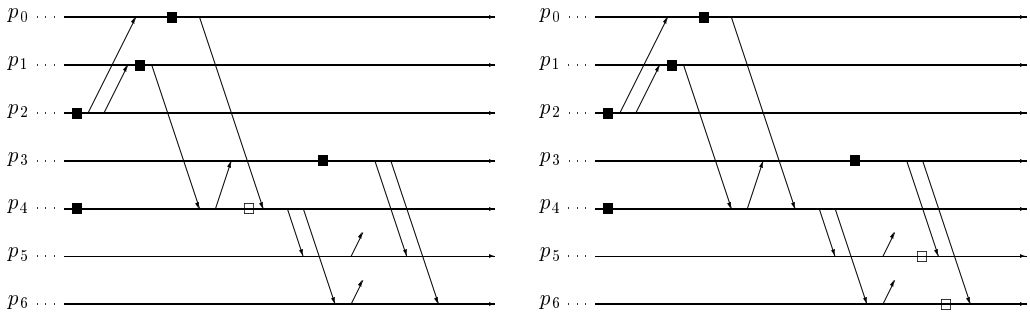


Figure 10: A stronger condition does not guarantee a better protocol

The use of a stronger condition does not necessarily mean that PRL will induce less checkpoints than BQC for every checkpoint pattern. Figure 10 shows a scenario in which p_4 takes a forced checkpoint only when BQC is considered; this situation is analogous to the one depicted in Figure 9 (b). From this point on the two checkpoints patterns diverge. The forced checkpoint taken by p_4 guarantees that no suspect Z-cycle will be detected by p_5 and p_6 (Figure 10 (a)), similar to the counter-example shown in Figure 7 (b). In Figure 10 (b), p_5 and p_6 must take forced checkpoints because they have received information from p_3 that a checkpoint in p_4 is obsolete.

A proof that a stronger condition does not necessarily lead to a better protocol was first obtained by Tsai, Kuo, and Wang for checkpointing protocols that satisfy the rollback-dependency trackability [20] and for index-based checkpointing protocols [21]. However, it is reasonable to expect that a protocol that uses a stronger condition will induce less checkpoints in most scenarios.

Simulation results Our experiments were conducted using the simulation environment provided by SPIN version 3.3.4 [10]. We have simulated a distributed computation considering the model described in Section 2. The communication pattern is uniform: any process can send (receive) messages to (from) any other process with equal probability. The probability of an internal event is the same as the probability of a communication event and basic checkpoints are taken every 8 internal events. The computation terminates when 500 basic checkpoints are taken. Figure 11 shows the number of induced checkpoints taken by both protocols with $2 \leq n \leq 14$; the results were collected from an average of 20 executions and using the same checkpoint patterns for both protocols.

The protocols start with same number of induced checkpoints for $n = 2$, but PRL takes consistently less checkpoints for $n > 2$ (Figure 11). The difference becomes less significant as the values of n are increased, because the situations where PRL can spare a checkpoint (Figure 9) becomes rarer.

Size of data structures An additional advantage of PRL is the total amount of control information that is maintained and propagated by the processes. BQC uses a vector clock and an $n \times n$ matrix of integers. However, as the diagonal of the matrix is not used by the

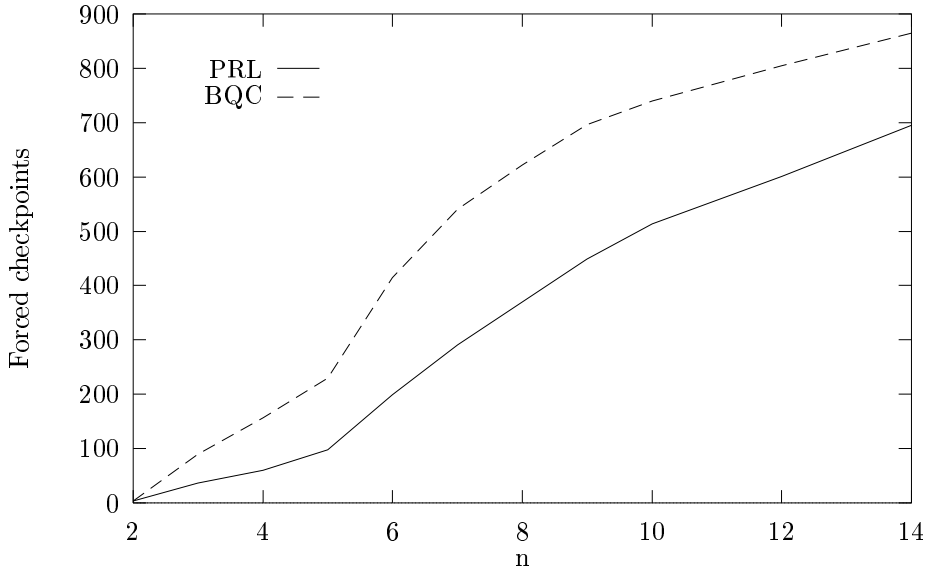


Figure 11: Simulations results: PRL and BQC

protocol, the vector clock can be embedded in the diagonal [2]. In contrast, PRL uses a vector clock and a vector of booleans. This represents a reduction from $O(n^2)$ to $O(n)$ in the complexity of the protocol.

BQC has been developed considering sequence of messages: zigzag paths and causal paths. For the description and proof of the protocol, the developers of BQC propose concatenation operators to construct causal and non-causal sequences of messages and checkpoints. For the development of PRL the authors took a different path, based upon Z-precedence [7]. The fact that it is formed by a composition of causal precedences eliminate the necessity of concatenation operators, probably reducing the complexity of the proof of Theorem 4.1.

In BQC, it is important to know in which checkpoint interval a process p_c sent a message to a process p_b. When all processes are taken into account, a matrix of integers is required. In PRL, we consider only causal precedences and not the exact intervals in which messages are sent. Instead, we keep track of the fact that a checkpoint $\hat{\sigma}_c^{\gamma-1}$ precedes *some* non-obsolete checkpoint $\hat{\sigma}_b^{\beta}$, lowering the control information required to a vector of booleans.

6 Conclusion

A recovery line is the most recent consistent global checkpoint from which a distributed computation can be restarted after a failure; it is also the last consistent global checkpoint of a progressive view [7]. In this paper, we have proposed a new quasi-synchronous checkpointing protocol (PRL) in which the processes propagate their local knowledge about the recovery line and collaborate to avoid domino effect [17]. PRL was designed taking

advantage of the theory developed for progressive views; in particular, we have explored Z-precedence, a higher-level interpretation of zigzag paths [16].

The literature presents many quasi-synchronous protocols that are domino-effect free [1, 2, 3, 4, 6, 9, 11, 14, 22]. Some of them enforce a stronger property, called rollback-dependency trackability (RDT), that states that all dependencies among checkpoints are causal dependencies [22]. Checkpointing protocols that enforce RDT usually require a vector clock [1, 11, 22]. In contrast, protocols that avoid the domino effect but do not enforce RDT are usually index-based protocols [3, 4, 9].

The protocol proposed by Baldoni, Quaglia, and Ciciani (BQC) [2] was the first protocol to use a vector clock and to avoid the domino effect without enforcing RDT. BQC maintains and propagates $O(n^2)$ control information. PRL enforces the same properties, but lowers the complexity of the required control information from $O(n^2)$ to $O(n)$. We have also presented theoretical and simulation evidences that PRL reduces the number of induced checkpoints in comparison with BQC.

At the moment, we are working on a comparison of PRL with index-based protocols. We already have initial simulation results comparing PRL to the protocol proposed by Briatico, Ciuffoletti, and Simoncini(BCS) [4]. These preliminary results evidence that the checkpoint strategy adopted to take basic checkpoints, the number of the processes in the distributed computation, and the communication pattern are parameters that affect which of the two protocols performs better. Additional experiments are required to fairly assess BCS and other index-based protocols against PRL.

Acknowledgments

This work has been supported by FAPESP under grant no. 99/01293-2 for Islene C. Garcia and grant no. 96/1532-9 for the Laboratory of Distributed Systems. Islene C. Garcia received financial support from CNPq under grant no. 145563/98-7. We also received financial support from PRONEX/FINEP, process no. 76.97.1022.00 (Advanced Information Systems).

We thank Gustavo M. D. Vieira for the model used in the simulation of the protocols.

References

[1] R. Baldoni, J. M. Helary, A. Mostefaoui, and M. Raynal. A communication-induced checkpoint protocol that ensures rollback-dependency trackability. In *IEEE Symp. on Fault Tolerant Computing (FTCS'97)*, 1997.

[2] R. Baldoni, F. Quaglia, and B. Ciciani. A VP-accordant checkpointing protocol preventing useless checkpoints. In *17-th Symp. on Reliable Distributed Systems (SRDS'98)*, Purdue University, 1998.

[3] R. Baldoni, F. Quaglia, and P. Fornara. An index-based checkpoint algorithm for autonomous distributed systems. *IEEE Trans. on Parallel and Distributed Systems*, 10(2), Feb. 1999.

[4] D. Briatico, A. Ciuffoletti, and L. Simoncini. A distributed domino-effect free recovery algorithm. In *IEEE 4th Symp. on Reliability in Distributed Software and Database Systems (SRDS'84)*, pages 207–215, 1984.

[5] M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Trans. on Computing Systems*, 3(1):63–75, Feb. 1985.

[6] E. N. Elnozahy, D. Johnson, and Y.M.Yang. A survey of rollback-recovery protocols in message-passing systems. Technical Report CMU-CS-96-181, School of Computer Science, Carnegie Mellon University, 1996.

[7] I. C. Garcia and L. E. Buzato. Progressive construction of consistent global checkpoints. In *19th IEEE Int. Conf. on Distributed Computing Systems (ICDCS'99)*, Austin, Texas, EUA, June 1999.

[8] J. Gosling, B. Joy, and G. L. Steele. *The Java Language Specification*. Java Series. Addison–Wesley, Sept. 1996.

[9] J.-M. Helary, A. Mostefaoui, R. Netzer, and M. Raynal. Preventing useless checkpoints in distributed computations. In *16th Symp. on Reliable Distributed Systems*, Durham, N. C., Oct. 1997.

[10] G. J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, 23(5), May 1997.

[11] T. R. K. Venkatesh and H. F. Li. Optimal checkpointing and local recording for domino-free rollback recovery. *Information Processing Letters*, 25(5):295–303, 1987.

[12] R. Koo and S. Toueg. Checkpointing and Rollback-Recovery for Distributed Systems. *IEEE Transactions on Software Engineering*, 13:23–31, jan 1987.

[13] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

[14] D. Manivannan and M. Singhal. Quasi-synchronous checkpointing: Models, characterization, and classification. *IEEE Trans. on Parallel and Distributed Systems*, 10(7), July 1999.

[15] F. Mattern. Virtual time and global states of distributed systems. In *Parallel and Distributed Algorithms*, pages 215–226. Elsevier Science Publishers B.V.(North-Holland), 1989.

[16] R. H. B. Netzer and J. Xu. Necessary and sufficient conditions for consistent global snapshots. *IEEE Trans. on Parallel and Distributed Systems*, 6(2):165–169, 1995.

[17] B. Randell. System structure for software fault tolerance. *IEEE Transactions on Software Engineering*, 1(2):220–232, June 1975.

[18] R.Baldoni, J.M.Helary, and M.Raynal. Rollback-dependency trackability: Visible characterizations. In *18th ACM Symposium on the Principles of Distributed Computing (PODC'99)*, Atlanta (USA), May 1999.

[19] Sun Microsystems Computer Corporation, Mountain View, CA, USA. *Java API Documentation Version 1.1*, Dec. 1996.

[20] J. Tsai, S. Y. Kuo, and Y. M. Wang. Theoretical analysis for communication-induced checkpointing protocols with rollback-dependency trackability. *IEEE Trans. on Parallel and Distributed Systems*, Oct. 1998.

[21] J. Tsai, S. Y. Kuo, and Y. M. Wang. Evaluations on domino-free communication-induced checkpointing protocols. *Information Processing Letters*, 69(1):31–37, Jan. 1999.

[22] Y. M. Wang. Consistent global checkpoints that contain a given set of local checkpoints. *IEEE Trans. on Computers*, 46(4):456–468, Apr. 1997.