A Linear Approach to Enforce the Minimal Characterization of the Rollback-Dependency Trackability Property

Islene C. Garvia    Luiz E. Buzato

Technical Report - IC-01-17 - Relatório Técnico

December - 2001 - Dezembro

The contents of this report are the sole responsibility of the authors.
O conteúdo do presente relatório é de única responsabilidade dos autores.
A Linear Approach to Enforce the
Minimal Characterization of the
Rollback-Dependency Trackability Property

Islene C. Garcia      Luiz E. Buzato
Instituto de Computação
Universidade Estadual de Campinas
Caixa Postal 6176
13083-970 Campinas, São Paulo, Brasil
{islene, buzato}@ic.unicamp.br

Abstract

A checkpointing protocol that enforces rollback-dependency trackability (RDT) during the progress of a distributed computation must take forced checkpoints to break non-trackable dependencies. Breaking just non-visibly doubled dependencies instead of breaking all non-trackable dependencies leads to fewer forced checkpoints, but seemed to require the processes of a computation to maintain and propagate $O(n^2)$ control information. In this paper, we prove that this hypothesis is false by presenting a protocol that breaks the minimal set of non-visibly doubled dependencies necessary to enforce RDT, called “non-visibly doubled PMM-paths”, using only $O(n)$ control information.

Keywords: fault-tolerance, rollback recovery, distributed checkpointing, distributed algorithms, algorithm complexity.

1 Introduction

A checkpointing protocol that enforces rollback-dependency trackability (RDT) during the progress of the computation must take forced checkpoints to break non-trackable dependencies [1, 12]. Although it is not possible to design an RDT protocol that will take the minimum number of forced checkpoints for all checkpoint and communication patterns [11], RDT protocols based on stronger induction conditions usually take fewer forced checkpoints than RDT protocols based on weaker conditions [3]. The inconvenience of the strongest approach, based on breaking only non-visibly doubled paths, was that it seemed to require $O(n^2)$ control information [3], while a weaker approach based on breaking all non-trackable

*This work is supported by FAPESP under grants no. 99/01293-2 (Ilsene C. Garcia), no. 96/1532-9 (Distributed Systems Laboratory), and no. 97/10982-0 (High Performance Computing Laboratory). Ilsene 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).
requires only $O(n)$ control information [12], where $n$ is the number of processes in the computation. The main contribution of this paper is to present a simple RDT protocol that implements the stronger approach in $O(n)$.

A checkpoint is a recording in stable memory of a process’ state that can be used for rollback recovery. The set of all checkpoints taken by a distributed computation and the dependencies established among these checkpoints due to message exchanges form a checkpoint and communication pattern (CCP). CCPs that satisfy RDT present only checkpoint dependencies that are on-line trackable using dependency vectors, and allow efficient solutions to the determination of the maximum and minimum consistent global checkpoints that include a set of checkpoints [12]. Many applications can benefit from these algorithms: rollback recovery, software error recovery, and distributed debugging [12].

Netzer and Xu have determined that checkpoint dependencies are created by sequences of messages called zigzag paths [10]. Two types of zigzag paths can be identified: causal paths (C-paths) and non-causal paths (Z-paths). C-paths are on-line trackable through the use of dependency vectors; Z-paths, on the contrary, cannot be on-line tracked. However, a CCP may present Z-paths and still satisfy RDT. In this case, all Z-paths must be doubled by a C-path; a Z-path is doubled by a causal one if the pair of checkpoints related by that Z-path is also related by a C-path [2, 3].

Baldoni, Helary and Raynal have established properties that could reduce the set of Z-paths that must be doubled in a CCP that satisfies RDT. They have concentrated their study on visible properties, that is, properties that can be tested on-line by an RDT protocol [2]. They have also proved that a process does not need to break a Z-path if it is able to detect that it is already causally doubled (a visibly doubled path). Additionally, they have conjectured that a specific set of Z-paths, named “non-visibly-doubled-EPSCM-paths”, determines the smallest set of Z-paths that must be tested for breaking by an RDT protocol [2]. Based on this set, they have proposed an RDT protocol that enforces this characterization using $O(n^2)$ control information, claiming that this protocol is optimal with respect to the size of control information [3].

Recently, we have proved that their conjecture was false and the set of Z-paths that must be tested for breaking by an RDT protocol can be further reduced to the set of “non-visibly-doubled-PMM-paths” [5]. In this paper, extending the approach used to prove the conjecture false, we describe a protocol that enforces this minimal characterization of RDT requiring only $O(n)$ control information.

This paper is structured as follows. Section 2 describes the computational model adopted. Section 3 introduces rollback-dependency trackability. Section 4 describes a quadratic approach to enforce the minimal characterization of RDT, similar to the one suggested by Baldoni, Helary, Mostefaoui, and Raynal [1, 3]. Section 5 presents a linear approach to enforce the minimal characterization of RDT. Section 6 concludes the paper.

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. The local history of a process \(p_i\) is modeled as a possibly infinite sequence of events \((e^1_i, e^2_i, \ldots)\), divided into internal events and communication events.

A checkpoint is an internal event that records the process’ state in stable memory. Each
process takes an initial checkpoint immediately after execution begins and a final
checkpoint immediately before execution ends. Let \(c^\gamma_i\) denote the \(\gamma\)th checkpoint taken
by \(p_i\). Two successive checkpoints \(c^{\gamma-1}_i\) and \(c^\gamma_i\), \(\gamma > 0\), define a
checkpoint interval \(I^\gamma_i\). An event \(e^\ell_i\) belongs to \(I^\gamma_i\) \((e^\ell_i \in I^\gamma_i)\) if it occurred in \(p_i\) after \(c^{\gamma-1}_i\), but not after \(c^\gamma_i\). Figure 1 illustrates a
checkpoint interval \(I^\gamma_i\) and an event \(e^\ell_i\) that belongs to \(I^\gamma_i\).

\[
\begin{align*}
\begin{array}{c}
p_i \ldots \quad c^{\gamma-1}_i \quad e^\ell_i \quad c^\gamma_i \\
I^\gamma_i
\end{array}
\end{align*}
\]

Figure 1: A checkpoint interval

The set of all checkpoints taken by a distributed computation and the dependencies
established among these checkpoints due to message exchanges form a
checkpoint and communication pattern (CCP). Figure 2 illustrates a CCP using a space-time diagram [9] augmented with checkpoints (black squares).

\[
\begin{align*}
\begin{array}{c}
p_0 \quad I^1_0 \quad I^2_0 \quad I^2_1 \quad I^3_0 \\
m_1 \quad m_2 \quad m_3
\end{array}
\end{align*}
\]

Figure 2: A distributed computation

2.1 Consistency

The concept of causal precedence is fundamental for a better understanding of consistency [9].

**Definition 2.1 Causal precedence**—Event \(e^\alpha_a\) causally precedes \(e^\beta_b\) \((e^\alpha_a \rightarrow e^\beta_b)\) if \((i) a = b\) and \(\beta = \alpha + 1\); \((ii) \exists m: e^\alpha_a = \text{send}(m)\) and \(e^\beta_b = \text{receive}(m)\); or \((iii) \exists e^\gamma_c: e^\alpha_a \rightarrow e^\gamma_c \land e^\gamma_c \rightarrow e^\beta_b\).

A cut of a distributed computation contains a prefix of each of the processes’ local histories. A consistent cut is left-closed under causal precedence and defines an instant in a distributed computation [4]. If a cut \(C \subset C’\), we can say that \(C\) is in the past of \(C’\) (Figure 2).
Definition 2.2 Consistent Cut — A cut C is consistent if, and only if,
\[ e \in C \land e' \rightarrow e \Rightarrow e' \in C \]

A consistent global state is formed by the states of each process in the frontiers of a consistent cut [4]. The set of consistent global checkpoints is a subset of the set of consistent global states. In Figure 2, C is related to a consistent global checkpoint, but C' is not.

2.2 Zigzag paths

Netzer and Xu have determined that checkpoints that are part of the same consistent global checkpoint cannot be related by sequences of messages called zigzag paths [10].

Definition 2.3 Zigzag path — A sequence of messages \( \mu = [m_1, \ldots, m_k] \) is a zigzag path from \( I_0^a \) to \( I_b^l \) if (i) \( p_a \) sends \( m_1 \) after \( c_0^a \); (ii) if \( m_i, 1 \leq i < k, \) is received by \( p_c \), then \( m_{i+1} \) is sent by \( p_c \) in the same or a later checkpoint interval; (iii) \( m_k \) is received by \( p_b \) before \( c_b^\beta \).

Two types of zigzag paths can be identified: (i) causal paths (C-paths) and (ii) non-causal paths (Z-paths). A zigzag path is causal if the reception of \( m_i, 1 \leq i < k, \) causally precedes the send event of \( m_{i+1} \). In Figure 2, \([m_1, m_2]\) is a C-path from \( I_0^1 \) to \( I_2^2 \) and \([m_3, m_4]\) is a Z-path from \( I_0^1 \) to \( I_2^2 \). A Z-path that starts in a checkpoint interval and finishes in a previous checkpoint interval of the same processes is a Z-cycle and identifies a useless checkpoint, that is, a checkpoint that cannot be part of any consistent global checkpoint [10]. Figure 3 presents a Z-cycle \([m_1, m_2, m_3]\) and a useless checkpoint \( c_i^\gamma \).

![Figure 3: A Z-cycle](image_url)

3 Rollback Dependency Trackability

The literature presents two approaches to define RDT. The first one is based on the study of on-line trackable dependencies, implemented through the use of dependency vectors [12]; the other one is based on the study of sequence of messages [1, 2, 3, 5].

3.1 Dependency vectors

A transitive dependency tracking mechanism can be used to capture causal dependencies among checkpoints. Each process maintains and propagates a size-\( n \) dependency vector. Let \( dv_i \) be the dependency vector of \( p_i, m, dv \) be the dependency vector piggybacked on a
message \( m \), and \( dv(c) \) be the dependency vector associated to a checkpoint \( c \). All entries of \( dv_i \) are initialized to 0. The entry \( dv_i[k] \) represents the current interval of \( p_i \) and it is incremented immediately after a checkpoint (including the initial one). Every other entry \( dv_i[j], j \neq i \), represents the highest interval index of \( p_j \) that \( p_i \) has knowledge about and it is updated using a component-wise maximum every time a message \( m \) with a greater value of \( m \), \( dv[j] \), arrives to \( p_i \). Figure 4 depicts the dependency vectors established during a distributed computation.

![Diagram](image)

**Figure 4:** A distributed computation with dependency vectors

Note in Figure 4 that \( dv(c^1) \) is \((1, 1, 1)\) and it correctly captures all zigzag paths that reach \( I^2_0 \). Unfortunately, not all dependencies can be tracked on-line. For example, \( dv(c^2) \) is \((1, 2, 2)\) and it does not capture the zigzag path from \( I^2_0 \) to \( I^2_2 \).

**Definition 3.1 On-line trackability**—A zigzag path from \( I^\alpha_a \) to \( I^\beta_b \) is on-line trackable through the use of dependency vectors if \( dv(c^\beta_b)[a] \geq \alpha \).

**Definition 3.2 Dependency vector characterization of RDT**—A checkpoint pattern satisfies RDT if all zigzag paths are on-line trackable.

RDT is a desirable property because efficient algorithms can be used to construct consistent global checkpoints if all zigzag paths are on-line trackable. Also, an RDT checkpoint pattern does not admit useless checkpoints [12].

### 3.2 Causal doubling

A CCP may present \( Z \)-paths and satisfy RDT if all \( Z \)-paths are doubled by a C-path [2, 3].

**Definition 3.3 Causal doubling**—A \( Z \)-path from \( I^\alpha_a \) to \( I^\beta_b \) is causally doubled if there is a C-path \( \mu \) from \( I^\alpha_a \) to \( I^\beta_b \) or \( a = b \) and \( \alpha \leq \beta \).

**Definition 3.4 Message-based characterization of RDT**—A checkpoint pattern satisfies RDT if all \( Z \)-paths are causally doubled.

A \( Z \)-path can be doubled by a causal one if the pair of checkpoints related by that \( Z \)-path is also related by a C-path [2, 3]. Another possibility for a \( Z \)-path from \( I^\alpha_a \) to \( I^\beta_b \) to
be doubled is if it starts and finishes in the same process and $I^\beta_b$ does not precede $I^\alpha_a$. In Figure 5 (a), $[m_1, m_2]$ is causally doubled by $m_3$ and in Figure 5 (b), $[m_1, m_2]$ is trivially doubled due to the execution of $p_a$.

\begin{center}
(a) $[m_1, m_2]$ is doubled by $m_3$
\end{center}

\begin{center}
(b) $[m_1, m_2]$ is trivially doubled
\end{center}

Figure 5: Causal doubling

3.3 RDT protocols

A communication-induced checkpointing protocol that enforces RDT allows processes to take checkpoints asynchronously, but they may be induced by the protocol to take forced checkpoints in order to break non-trackable dependencies [1, 12]. Forced checkpoints must be taken upon the arrival of a message, but before this message is processed by the computation. The decision to take a forced checkpoint must be based only on the local knowledge of a process; there are no control messages, no global knowledge or knowledge about the future of the computation. These assumptions impose some restrictions on the set of CCPs that can be produced by RDT protocols.

The CCP depicted in Figure 6 (a), for example, would never have been produced by an RDT protocol. This pattern has a Z-path $[m_1, m_2]$ that is doubled by message $m_3$ in the future of a consistent cut $\mathcal{C}$. At $\mathcal{C}$, the processes of the computation cannot rely on the existence of $m_3$, since a scenario such as the one depicted in Figure 6 (b) could have happened, producing a CCP that does not satisfy RDT. Under an RDT protocol, the CCP presented in Figure 6 (a) should present at least one forced checkpoint (Figure 6 (c)).

\begin{center}
(a)\hspace{2cm} (b)\hspace{2cm} (c)
\end{center}

\begin{center}
\begin{tabular}{c}
\includegraphics[width=0.3\textwidth]{figure6a.png} \\
\includegraphics[width=0.3\textwidth]{figure6b.png} \\
\includegraphics[width=0.3\textwidth]{figure6c.png}
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{c}
\scriptsize{\textbf{Figure 6:} The behavior of RDT protocols}
\end{tabular}
\end{center}

The RDT property must be enforced in every consistent cut of a computation that runs an RDT protocol. This observation has led us to introduce the concept of left-doubling [5].
3.4 Left-doubling

A C-path $\mu$ belongs to a consistent cut $\mathcal{C}$ if the reception of the last message of $\mu$ belongs to $\mathcal{C}$. A Z-path $\zeta$ can be seen as a concatenation of $\ell$ C-paths $\mu_1 \cdot \mu_2 \cdot \ldots \cdot \mu_\ell$ and $\zeta$ belongs to a consistent cut $\mathcal{C}$ if all causal components of $\zeta$ belong to $\mathcal{C}$.

**Definition 3.5 Left-doubling**—A Z-path $\zeta$ is left-doubled in relation to a consistent cut $\mathcal{C}$ if (i) $\zeta$ belongs to $\mathcal{C}$ and (ii) $\zeta$ is doubled by a C-path $\mu$ that also belongs to $\mathcal{C}$.

A consistent cut $\mathcal{C}$ satisfies the RDT property if, and only if, all Z-paths that belong to $\mathcal{C}$ are left-doubled. Using the concept of left-doubling, we have proved that a protocol that breaks all “non-visibly doubled PMM-paths” must enforce RDT [5].

3.5 The minimal characterization of RDT

**Definition 3.6 PMM-path**—A PMM-path is a Z-path composed of a prime $m$essage $m_1$ and a $m$essage $m_2$.

A message $m$ from $I_k^c$ to $p_k$ is prime if $m$ is the first message received by $p_k$ that brings information about $I_k^c$. Figure 7 presents a PMM-path $[m_1] \cdot [m_2]$ from $I_k^c$ to $I_j^c$.

![Figure 7: A PMM-path](image)

**Definition 3.7 Visibly Doubled PMM-path**—A PMM-path $[m_1] \cdot [m_2]$ is visibly doubled if (i) is causally doubled by a C-path $\mu$ and (ii) the reception of the last message of $\mu$ causally precedes the sending of $m_1$.

![Figure 8: A visibley doubled PMM-path](image)
Figure 8 presents a visibly doubled PMM-path \([m_1] \cdot [m_2]\) from \(I^k_i\) to \(I^j_j\). We should note that \([m_1] \cdot [m_2]\) is left-doubled in relation to any consistent cut of the computation, since any consistent cut that contains \(m_1\) should also contain \(\mu\). The set of "non-visibly-doubled PMM-paths" characterizes the minimal set of Z-paths that be must tested for breaking by an RDT protocol [5].

**Definition 3.8** The minimal characterization of RDT—A CCP satisfies the RDT property if all PMM-paths are visibly doubled.

In the following sections, we are going to focus on the problem of implementing an RDT protocol that enforces this minimal characterization.

### 4 A quadratic approach

The core of a protocol that enforces the minimal characterization of RDT lies on the detection of non-visibly doubled PMM-paths by a process \(p_i\). Let us consider a PMM-path \([m_1] \cdot [m_2]\) from \(I^k_i\) to \(I^j_j\) such that \(m_1\) is received by \(p_k\) after the sending of \(m_2\) (Figure 7). Before processing \(m_1\), \(p_k\) must detect the establishment of this PMM-path and verify whether it is visibly doubled (Figure 8). If \([m_1] \cdot [m_2]\) is visibly doubled, \(m_1\) can be processed immediately. Otherwise, \(p_i\) must take a forced checkpoint before processing \(m_1\).

#### 4.1 Detecting PMM-paths

In order to detect all PMM-paths formed upon the reception of a message, process \(p_i\) must record for what processes it has sent messages during the current interval. To do this, \(p_i\) maintains a vector of booleans \(sent \_jo_i\), such that all entries of \(sent \_jo_i\) are set to \texttt{false} when \(p_i\) takes a checkpoint, and an entry \(sent \_jo_i[j]\) is set to \texttt{true} when \(p_i\) sends a message to \(p_j\). A PMM-path is detected by \(p_i\) upon the reception of a message \(m\) from \(p_k\) when the following condition holds:

\[
\exists j : sent \_jo_i[j] \land m \_dv[k] > dv_i[k]
\]

#### 4.2 Detecting non-visibly doubled PMM-paths

The detection of whether \([m_1] \cdot [m_2]\) from \(I^k_i\) to \(I^j_j\) is visibly doubled by a C-path \(\mu\) can be divided into two cases:

**I. From the point of view of \(p_k\), the interval \(I^j_j\) is in the past of \(p_j\)**

Figure 9 shows a scenario in which \(p_i\) receives knowledge that \(m_2\) was received during \(I^j_j\), but \(\mu\) was received during \(I^j_{j+1}\). Since a C-path \(\mu\) from \(I^k_i\) to \(I^j_{j+1}\) does not double a PMM-path from \(I^k_i\) to \(I^j_j\), process \(p_i\) must take a forced checkpoint before processing \(m_1\).

To detect visibly doubled PMM-paths, \(p_i\) must evaluate (i) whether \(m_2\) was received by \(p_j\) and in which checkpoint interval, and (ii) whether \(p_j\) has received knowledge about \(I^k_i\) and in which checkpoint interval. For \(p_i\) to be able to answer these questions, the processes of the computation would have to maintain and propagate an unbounded amount of control information, proportional to the number of messages and checkpoint intervals.
II. From the point of view of $p_i$, the interval $I_j^γ$ is not in the past of $p_j$
Figure 10 presents three scenarios to show that the problem of detecting visibly doubled paths is much easier when in $p_i$’s view $I_j^γ$ is not in the past of $p_j$. In Figure 10 (a), $p_i$ receives knowledge that both $m_2$ and $μ$ were received during $I_j^γ$. In Figures 10 (b, c), $p_i$ receives knowledge that $μ$ was received by $p_j$, but $p_i$ does not receive knowledge about the reception of $m_2$. In these cases, the existence of a C-path $μ$ from $I_k^γ$ to $p_j$ guarantees to $p_i$ that $[m_1] \cdot [m_2]$ is causally doubled.

Figure 10: $I_j^γ$ is not in the past of $p_j$

Fortunately, there is an approach to handle case I that requires only $O(n)$ control information, as explained in next Section. Section 4.4 shows an $O(n^2)$ approach to handle case II. In Section 5, we are going to prove that case II can also be handled in $O(n)$.

4.3 Process $p_i$ knows that $I_j^γ$ is in the past

Let us assume that, upon the reception of $m_1$, $p_i$ knows that $m_2$ was received in an interval that is on the past of $p_j$. In Figure 11 (a), $p_i$ receives knowledge about $I_j^{γ+1}$ due a C-path $ν$ that arrives to $p_i$ before $m_1$. The concatenation of $ν$ and $m_2$ forms a Z-cycle. Since
the RDT property does not allow Z-cycles, \( p_i \) should have taken a forced checkpoint before processing the last message of \( \nu \). The information about \( I_j^{\gamma+1} \) could have arrived with \( m_1 \). In Figure 11 (b), \( p_k \) receives knowledge about \( I_j^{\gamma+1} \) due a C-path \( \nu \) that arrives to \( p_k \) before the sending of \( m_1 \). The concatenation of \( \nu, m_1 \) and \( m_2 \) also forms a Z-cycle and \( p_k \) should have taken a forced checkpoint before processing \( m_1 \).

![Diagram of Z-cycles](image)

**Figure 11:** Process \( p_k \) knows that \( I_j^{\gamma} \) is in the past

The Z-cycle \([\nu] \cdot [m_2] \) of Figure 11 (a) and the Z-cycle \([\nu \cdot m_1] \cdot [m_2] \) of Figure 11 have only two causal components. Z-cycles formed by two causal components \([\nu_1] \cdot [\nu_2] \) and are called CC-cycles (Figure 12). CC-cycles can be easily detected and broken on-line if \( p_i \) takes a forced checkpoint before processing the last message of \( \nu_1 \). Process \( p_k \) must take the forced checkpoint only if \([\nu_2] : [\nu_1] \) “contains” a checkpoint, that is, it is not simple \([1, 3]\).

![Diagram of CC-cycles](image)

**Figure 12:** A CC-cycle \([\nu_1] \cdot [\nu_2] \)

To keep track of simple paths, each process maintains and propagates a size-\( n \) boolean vector simple. Let simple\(_i\) be the vector maintained by \( p_i \), and m.simple be the vector piggybacked on a message \( m \). The entry simple\(_i\)[k] is always \texttt{true}, and the entries simple\(_i\)[k], \( k \neq i \), are reset to \texttt{false} when \( p_i \) takes a checkpoint. When \( p_i \) receives a message \( m \), each entry simple\(_i\)[k] is updated as follows:

\[
\text{if } m.\text{dv}[k] > \text{dv}_i[k] \text{ then } \text{simple}_i[k] \leftarrow m.\text{simple}[k] \\
\text{if } m.\text{dv}[k] = \text{dv}_i[k] \text{ then } \text{simple}_i[k] \leftarrow \text{simple}[k] \land m.\text{simple}[k]
\]

Process \( p_k \) detects a CC-cycle upon the reception of \( m \) using the following condition:

\[
m.\text{dv}[i] = \text{dv}_i[i] \land m.\text{simple}[i] = \texttt{false}
\]

In the scenarios of Figure 11, the second causal component of the CC-cycle is represented by a single message \( m_2 \). However, keeping track of only CC-cycles of the form \([\nu]m \) would increase the complexity of the required control information due to the propagation
of knowledge about single messages. Also, as an RDT protocol must break all CC-cycles, using the above condition does not increase the number of forced checkpoints.

4.4 Tracking C-paths from $I_k$ to $p_j$

Even if a process $p_i$ breaks CC-cycles, PMM-paths such as the ones described in case II of Section 4.2 (Figure 10) needed to be tested for breaking. Thus, if $p_i$ has sent a message $m_2$ to $p_j$ and receives a prime message $m_1$ from $I_k^c$, $p_i$ must verify whether there is a C-path $\mu$ from $I_k^c$ to $p_j$. We should note that before the reception of $m_1$, process $p_i$ cannot have information about $\mu$, otherwise $m_1$ would not be prime (Figure 13). Thus, the information about $\mu$ can only arrive on the control information piggybacked on $m_1$.

$$
\begin{array}{c}
p_j \ldots \quad I_j^c \quad m_2 \\
p_i \ldots \\
p_k \ldots \quad I_k^c \quad \mu \\
\end{array}
$$

Figure 13: Message $m_1$ is not prime

According to the definition of dependency vectors, $dv_i[k]$ is the greatest interval index of $p_k$ that $p_i$ has knowledge about. Let us consider a matrix of booleans $\text{causal}_i$ such that each entry $\text{causal}_i[k][j]$ indicates whether, up to the knowledge of $p_k$, there is a C-path from $I_k^{dv_i[k]}$ to $p_j$. The entries on the diagonal of $\text{causal}_i$ are always true, since there is a trivial causal flow from a process to itself. Every other entry $\text{causal}_i[k][j], k \neq j$ is initialized to false. When $p_k$ takes a checkpoint, all entries $\text{causal}_i[i][j], i \neq j$ are reset to false, indicating that there is no C-path from this new interval. Let $m, \text{causal}$ be a matrix piggybacked on a message $m$. When $p_k$ receives $m$ from $p_i$, $\text{causal}_i$ is updated as follows:

$$\forall k, \text{if } m, dv[k] > dv_i[k] \text{ then } \forall l : \text{causal}_i[k][l] \leftarrow m, \text{causal}[k][l]$$

$$\forall k, \text{if } m, dv[k] = dv_i[k] \text{ then } \forall l : \text{causal}_i[k][l] \leftarrow \text{causal}_i[k][l] \lor m, \text{causal}[k][l]$$

$$\text{causal}_i[s][i] \leftarrow \text{true}$$

$$\forall l : \text{causal}_i[l][i] \leftarrow \text{causal}_i[l][i] \lor \text{causal}_i[l][s]$$

4.5 Checkpoint induction condition

A forced checkpoint is induced by $p_k$ upon the reception of a prime message $m$ from $I_k^c$ if (i) there is a CC-cycle or (ii) there is a PMM-path from $p_k$ to $p_j$, but there is no C-path from $I_k^c$ to $p_j$:

(i) \((m, dv[i] = dv_i[i] \land m, \text{simple}[i] = \text{false}) \lor \)

(ii) \((\exists j : \text{sent}_j, o_i[j] \land m, dv[k] > dv_i[k] \land \neg m, \text{causal}_i[k][j])\)
The approach presented in this Section is similar to the one presented by Baldoni, Helary, Mostefaoui and Raynal, although their protocols break more complex Z-paths [1, 3]. Their approach requires $O(n^2)$ control information since it tracks the existence of C-paths from every process $p_k$ to every process $p_j$ of the computation. Baldoni, Helary, and Raynal claim that $O(n^2)$ is optimal with respect to the size of the control information [3]. In the next Section, we are going to show an $O(n)$ RDT protocol that breaks visibly-doubled Z-paths.

5 A linear approach

Linearity of the control information comes as a result of two observations: (i) we do not need to keep track of C-paths from $I^k$ to $p_j$, instead, we are going to take advantage of dependency vector restrictions imposed by an RDT protocol and (ii) we can perform comparison operations on dependency vectors as a whole instead of keeping track of single entries, as in Definition 3.1. This alternative approach is the key to the complexity reduction. Thus, let us define the following comparison operations:

$$ dv(c^\gamma_j) \geq dv(c^\kappa_k) \iff \forall i, 0 \leq i < N, \ dv(c^\gamma_j)[i] \geq dv(c^\kappa_k)[i] $$
$$ dv(c^\gamma_j) = dv(c^\kappa_k) \iff \forall i, 0 \leq i < N, \ dv(c^\gamma_j)[i] = dv(c^\kappa_k)[i] $$

5.1 Dependency vector restrictions under RDT

Let us begin with dependency vector restrictions that must hold for all CCPs, not only on CCPs produced by RDT protocols.

**Theorem 5.1** Under RDT, the existence of a C-path $\mu$ from $I^k$ to $I^\gamma_j$ guarantees that $dv(c^\gamma_j) \geq dv(c^\kappa_k)$.

**Proof:** For the sake of contradiction, let us assume the existence of an entry $l$ of $dv(c^\gamma_j)$ such that $dv(c^\gamma_j)[l] < dv(c^\kappa_k)[l] = \lambda$ (Figure 14). The information about $I^\kappa_k$ must have arrived at $p_k$ due to a C-path $\mu'$ and the last message of $\mu'$ must have been received after the sending of the first message of $\mu$. The concatenation of $\mu$ and $\mu'$ forms a Z-path from $I^\kappa_k$ to $I^\gamma_j$ that is not on-line trackable (a non-causally doubled Z-path).

![Figure 14: Contradiction hypothesis of Theorem 5.1](image)

---

Islène C. García and Luíz E. Buzato
Corollary 5.2 Under RDT, the existence of a C-path \( \mu \) from \( I_k^\mu \) to \( I_j^\gamma \) and of a C-path \( \mu' \) from \( I_j^\gamma \) to \( I_k^\kappa \) guarantees that \( dv(e_j^\gamma) = dv(e_k^\kappa) \).

Proof: Due to \( \mu \), \( dv(e_j^\gamma) \geq dv(e_k^\kappa) \). Due to \( \mu' \), \( dv(e_k^\kappa) \geq dv(e_j^\gamma) \). Thus, \( dv(e_j^\gamma) = dv(e_k^\kappa) \). \( \square \)

\[ \begin{array}{c}
 p_j \cdots \xrightarrow{e_j^\gamma} I_j^\gamma \xrightarrow{\mu} e_k^\kappa \xrightarrow{\mu'} I_k^\kappa \cdots \nabla \end{array} \]

Figure 15: The existence of \( \mu \) and \( \mu' \) guarantees that \( dv(e_j^\gamma) = dv(e_k^\kappa) \)

5.2 Dependency vector restrictions under an RDT protocol

Since an RDT protocol must enforce RDT in every consistent cut of the computation, let us explore dependency vector restrictions during the progress of checkpoint intervals.

Theorem 5.3 Let \( \mu \) be a C-path \( \mu \) from \( I_k^\mu \) to \( I_j^\gamma \) and let \( C \) be a consistent cut that contains \( \mu \). Let \( e_j^\gamma \in I_j^\gamma \) and \( e_k^\kappa \in I_k^\kappa \) be the events of \( p_j \) and \( p_k \) that belong to the frontier of \( C \).

Under an RDT protocol, the following restriction should hold: \( dv(e_j^\gamma) \geq dv(e_k^\kappa) \).

Proof: For the sake of contradiction, let us assume the existence of an entry \( l \) of \( dv(e_j^\gamma) \) such that \( dv(e_j^\gamma)[l] < dv(e_k^\kappa)[l] = \lambda \) (Figure 16). The information about \( I_k^\lambda \) must have arrived at \( p_k \) due to a C-path \( \mu' \) and the last message of \( \mu' \) must have been received after the sending of the first message of \( \mu \). The concatenation of \( \mu \) and \( \mu' \) forms a Z-path from \( I_k^\lambda \) to \( I_j^\gamma \) that is not left-doubled in relation to \( C \). \( \square \)

\[ \begin{array}{c}
 p_j \cdots \xrightarrow{e_j^\gamma} I_j^\gamma \xrightarrow{\mu} e_k^\kappa \xrightarrow{\mu'} I_k^\kappa \cdots \nabla \end{array} \]

Figure 16: Contradiction hypothesis of Theorem 5.3.

Corollary 5.4 Let \( \mu \) be a C-path \( \mu \) from \( I_k^\mu \) to \( I_j^\gamma \) and \( \mu' \) be a C-path from \( I_j^\gamma \) to \( I_k^\kappa \). Let \( C \) be a consistent cut that contains \( \mu \) and \( \mu' \). Let \( e_j^\gamma \in I_j^\gamma \) and \( e_k^\kappa \in I_k^\kappa \) be the events of \( p_j \) and \( p_k \) that belong to the frontier of \( C \).

Under an RDT protocol, the following restriction should hold: \( dv(e_j^\gamma) = dv(e_k^\kappa) \).

\[ \begin{array}{c}
 p_j \cdots \xrightarrow{e_j^\gamma} I_j^\gamma \xrightarrow{\mu} e_k^\kappa \xrightarrow{\mu'} I_k^\kappa \cdots \nabla \end{array} \]
\textbf{Proof:} Due to $\mu$, $dv(e_j^\gamma) \geq dv(e_k^\kappa)$. Due to $\mu'$, $dv(e_j'^\gamma) \geq dv(e_k'^\kappa)$. Thus, $dv(e_j^\gamma) = dv(e_k'^\kappa)$. \hfill \square

\begin{figure}[h]
\centering
\begin{tikzpicture}[> = stealth, auto, scale = 0.8]
\node (1) at (0,0) {$p_j \cdots I_j^\gamma$};
\node (2) at (1,0) {$e_j^\gamma$};
\node (3) at (2,0) {$p_k \cdots I_k^\kappa$};
\node (4) at (2,1) {$\mu'$};
\node (5) at (2,2) {$e_k'^\kappa$};
\node (6) at (2,3) {$c$};
\draw[-] (1) -- (2);
\draw[-] (3) -- (5);
\draw[-] (4) -- (6);
\end{tikzpicture}
\caption{The existence of $\mu$ and $\mu'$ guarantees that $dv(e_j^\gamma) = dv(e_k'^\kappa)$}
\end{figure}

5.3 Process $p_k$ knows that $dv_j = dv_k$

Let $\mu$ be a C-path from $I_j^\gamma$ to $I_j^\gamma$ and $\mu'$ be a C-path from $I_j^\gamma$ to $I_k^\kappa$ such that the last message of $\mu$ is received before the first message of $\mu'$ is sent (Figure 18). Upon the arrival of $\mu'$, $p_k$ receives knowledge about $\mu$ and, according to Corollary 5.4, it is also able to conclude that $dv_j = dv_k$.

\begin{figure}[h]
\centering
\begin{tikzpicture}[> = stealth, auto, scale = 0.8]
\node (1) at (0,0) {$p_j \cdots I_j^\gamma$};
\node (2) at (1,0) {$\mu'$};
\node (3) at (2,0) {$p_k \cdots I_k^\kappa$};
\node (4) at (2,1) {$e_k'^\kappa$};
\node (5) at (2,2) {$c$};
\draw[-] (1) -- (2);
\draw[-] (3) -- (4);
\draw[-] (2) -- (5);
\end{tikzpicture}
\caption{Process $p_k$ knows that $dv_j = dv_k$}
\end{figure}

The following theorem shows that the verification of equal dependency vectors can replace the verification of the existence of C-paths.

\textbf{Theorem 5.5} Let $I_k^\kappa$ be the current interval of a process $p_k$. Under an RDT protocol, $p_k$ knows the existence of a C-path $\mu$ from $I_k^\kappa$ to $p_j$ if, and only if, to the knowledge of $p_k$, $dv_j = dv_k$.

\textbf{Proof:}
(i) $dv_j = dv_k$ $\Rightarrow$ a C-path $\mu$ from $I_k^\kappa$ to $p_j$

Since $dv_j = dv_k$, $dv_j[k] = dv_k[k] = \kappa$ and there must be a C-path $\mu$ from $I_k^\kappa$ to $p_j$.

(ii) a C-path $\mu$ from $I_k^\kappa$ to $p_j$ $\Rightarrow$ $dv_j = dv_k$

Process $p_k$ must have received knowledge about $\mu$ due to a C-path $\mu'$ from $p_j$ to $p_k$. The first message of $\mu'$ must have been sent after the reception of the last of $\mu$. Since an RDT protocol does not allow CC-cycles, there cannot be a checkpoint between the reception of the last of $\mu$ and the sending of first message of $\mu'$. Thus, these two events occurred in the same checkpoint interval and, according to Corollary 5.4, upon the reception of the last message of $\mu'$, $p_k$ knows that $dv_j = dv_k$ (Figure 18).

Also, $p_k$ will not be able to increase its dependency vector till the end of $I_k^\lambda$. For the sake of contradiction, let us assume that $p_k$ receives information about $I_k^\lambda$ through a C-path
\( \nu \) after the reception of the last message of \( \mu' \). According to Corollary 5.2, \( dv(c^\lambda_j) \) should be equal to \( dv(c^\lambda_k) \), and \( p_j \) must also receive information about \( I^\lambda_i \) through a C-path \( \nu' \). Let \( C \) be the minimum consistent cut that contains \( \mu' \), that is, the cut formed by the reception of the last message of \( \mu' \) and all the events that causally precede this reception (Figure 19). Thus, at \( C \), neither \( p_j \) nor \( p_k \) have knowledge about \( I^\lambda_i \). From \( C \) it is possible to construct a sequence of consistent cuts that reflect the progress of the computation, adding one event at a time. Either \( \nu \) or \( \nu' \) is going to be included first during the sequence. If \( \nu \) is included first, we would have a consistent cut, say \( C' \), such that \( [\nu] \cdot [\mu] \) is not left-doubled in relation to \( C' \) (Figure 19). Analogously, if \( \nu' \) is included first, we would have a consistent cut such that \( [\nu'] \cdot [\mu'] \) is not left-doubled in relation to it. \( \square \)

![Figure 19: Contradiction hypothesis of Theorem 5.5](image)

**Corollary 5.6** Under an RDT protocol, \( p_k \) knows that \( dv_k = dv_j \) if \( dv_k[k] = dv_j[k] \).

**Proof:** Since \( dv_k[k] = dv_j[k] \), there must exist a C-path from the current interval of \( p_k \) to \( p_j \) and, according to Theorem 5.5, \( dv_k = dv_j \). \( \square \)

### 5.4 Keeping track of equal dependency vectors

Each process maintains and propagates a size-\( n \) boolean vector \( equal \). Let \( equal_i \) be the vector maintained by \( p_i \), and \( m.\ equal \) be the vector piggybacked on a message \( m \). The entry \( equal_i[k] \) is always true, and the entries \( equal_i[k], k \neq i \), are reset to false when \( p_i \) takes a checkpoint. When \( p_i \) receives a message \( m \) from \( p_k \) without taking a forced checkpoint, it must update \( equal_i \). If \( m.dv[i] = dv_i[i] \), \( p_i \) learns that \( dv_i = dv_k \) (Figure 20 (a)). If up to the knowledge of \( p_k \) when it sent \( m \) there is a process \( p_j \) such that \( dv_k = dv_j \), \( p_i \) also learns that \( dv_i = dv_j \) (Figure 20 (b)). This behavior can be summarized as follows:

\[
\text{if } m.dv[i] = dv_i[i] \text{ then } \forall j : equal_i[j] \leftarrow equal_i[j] \lor m.equal[j]
\]

There is no need for \( p_k \) to propagate additional information about dependency vectors, because if, up to the knowledge of \( p_k \), \( dv_j \neq dv_k \), \( p_k \) cannot derive from any information contained in \( m \) that \( dv_i = dv_j \). According to Corollary 5.6, to the knowledge of \( p_k \), \( dv_j[k] \neq dv_k[k] \). When \( p_i \) receives \( m \), \( dv_i[k] = dv_k[k] \neq dv_j[k] \) and \( dv_i \neq dv_j \) (Figure 20 (c)). Thus, keeping track of equal dependency vectors requires only \( O(n) \) control information.
5.5 Checkpoint induction condition

A forced checkpoint is induced by \( p_i \) upon the reception of a prime message \( m \) from \( p_k \) if (i) there is a CC-cycle or (ii) there is a PMM-path from \( p_k \) to \( p_j \), but \( dv_k \neq dv_j \):

\[
\begin{align*}
(i) & \quad (m, dv[i] = dv_i[i] \land m, simple[i] = false) \lor \\
(ii) & \quad (\exists j: sent_m, o_i[j] \land m, dv[k] > dv_i[k] \land \neg m, causal[j])
\end{align*}
\]

We should note the above checkpoint induction condition is analogous to the one presented in Section 4.5. The only difference is the replacement of the test \( \neg m, causal[k][j] \) for \( \neg m, causal[j] \).

5.6 Optimizations

Let us continue to explore properties of the processes' behavior under an RDT protocol to simplify an implementation of the minimal characterization of RDT.

**Theorem 5.7** If \( p_i \) receives a non-prime message \( m \), all entries of \( m, dv \) are known by \( p_k \).

**Proof:** Assume that \( m \) was sent by \( p_k \) during \( I^k \), and there is a C-path \( \mu \) from \( I^k \) to \( p_i \) such that \( \mu \) arrived to \( p_i \) before \( m \). Assume that \( m, dv[l] = \lambda \) and let \( \mu' \) be a C-path from \( p_k \) to \( p_k \) that arrived to \( p_k \) after the sending of the first message of \( \mu \) and before the sending of \( m \). It is possible to construct a consistent cut \( \mathcal{C} \) such that \( \mu', \mu \) is not left-doubled in relation to \( \mathcal{C} \) (Figure 21). \( \square \)

![Diagram](image-url)
Due to Theorem 5.7, upon the reception of a non-prime message $m_i$, there is no need to check and update $dv_i$. Also, since an entry of $simple_i$ can change only if at least one entry of $dv_i$ has changed, the updating of $simple_i$ can be skipped.

According to the second part of the proof of Theorem 5.5, when $p_k$ knows that $p_j$ knows its current interval, say $I_k^c$, $p_k$ cannot increase $dv_k$ till the end of $I_k^c$. Thus, we can divide a checkpoint interval of any process $p_k$ into three phases:

**Phase 0:** while no message has been sent, no PMM-path can be formed, and $dv_i$ can incorporate new dependencies without restrictions.

**Phase 1:** after at least one message has been sent, $dv_i$ can change according to the induction condition presented in Section 5.5.

**Phase 2:** after $p_k$ has received knowledge about other process with an equal dependency vector, no new dependency can be incorporated into $dv_i$.

Unfortunately, the updating of vector $equal$ cannot benefit from the described optimizations. Figure 22 illustrates that vector $equal$ must be updated even if no new dependency is established. When $p_1$ receives the second message from $p_0$, it does not change $dv_1$. However, $p_1$ receives knowledge that $dv_1 = dv_0$. Using this information, $p_2$ will be able to save a forced checkpoint when it receives a message from $p_1$.

![Figure 22: Vector $equal$ must be updated even if no new dependency is established](image)

Figure 23 illustrates that vector $equal$ must be updated during phase 2. Process $p_2$ starts phase 2 when it receives a message from $p_1$ and it learns that $dv_1 = dv_2$. When $p_0$ receives a message from $p_1$, it learns that $dv_0 = dv_1$. Also, when $p_0$ sends a message to $p_2$, $p_2$ learns that $dv_0 = dv_2$. Using this information $p_3$ will be able to save a forced checkpoint when it receives a message from $p_2$. Thus, even if a process is in the phase 2 of the algorithm, it must continue to update vector $equal$ because the collected information may help other processes to save checkpoints.

![Figure 23: Vector $equal$ must be updated during phase 2](image)
An implementation of the minimal characterization of RDT including all optimizations is described in Class RDT_Minimal, using Java\(^1\) [7].

```java
public class RDT_Minimal {
    public static int N = 100; // Number of processes in the application
    public int pid;            // Unique process' identifier
    protected int [] dv = new int[N]; // Dependency vector, automatically initialized to (0,...,0)
    protected boolean [] equal = new boolean [N]; // Keeps track of equal dependency vectors
    protected boolean [] simple = new boolean [N]; // Keeps track of simple paths
    protected boolean [] sent_to = new boolean [N]; // Keeps track of sent messages
    public int phase;

    public class Message {
        public int sender, receiver;
        public int [] dv;
        public boolean [] equal;
        public boolean [] simple;
    }

    public void takeCheckpoint () {
        // Write state into stable memory
        for (int i=0; i < N; i++) {
            equal[i] = false;
            simple[i] = false;
            sent_to[i] = false;
        }
        equal[pid] = true;
        simple[pid] = true;
        dv[pid]++;
        phase = 0;
    }

    public RDT_Minimal(int pid) { this.pid = pid; } // Constructor

    public void run () { takeCheckpoint(); } // Start execution

    public void finalize() { takeCheckpoint(); } // Finish execution

    public void sendMessage(Message m) {
        m.dv = (int []) dv.clone(); // Piggybacks control information
        m.equal = (boolean []) equal.clone();
        m.simple = (boolean []) simple.clone();
        sent_to[m.receiver] = true;
        if (phase == 0) phase = 1;
        // Send message
    }
}

\(^1\)Java is a trademark of Sun Microsystems, Inc.
```
Class RDT_Minimal.java

```java
private boolean mustTakeForcedCheckpoint(Message m) {
    if (phase == 0) return false;   // Every new dependency can be accepted
    if (phase == 2) return true;    // No new dependency can be accepted
    if (m.dv[pid] == dv[pid] && !m.simple[pid]) return true; // Non causally doubled CC-cycle
    for (int i=0; i < N; i++) {     // Verify whether all PMM-paths are visibly doubled
        if (sent[i] && !m.equal[i]) return true;
        return false;
    }
}

public void receiveMessage(Message m) {
    if (m.dv[m.sender] > dv[m.sender]) {   // New dependency
        takeCheckpoint();
    }
    if (mustTakeForcedCheckpoint(m)) {   // Dependency vector update
        for (int i=0; i < N; i++)
            if (m.dv[i] > dv[i]) {
                dv[i] = m.dv[i];
                simple[i] = m.simple[i];
            } else if (m.dv[i] == dv[i])
                simple[i] = simple[i] && m.simple[i];
            }
    if (m.dv[pid] == dv[pid]) {   // m.dv == dv
        for (int i=0; i < N; i++)
            equal[i] = equal[i] || m.equal[i];
        phase = 2;
    }
    // Message is processed by the application
}
```

6 Conclusion

The simplest RDT protocols are based only on checkpoints, message-send, and message-receive events: No-Receive-After-Send, Checkpoint-After-Send, Checkpoint-Before-Receive, and Checkpoint-After-Send-Before-Receive [12]. Clearly, these protocols are prone to induce a very large number of forced checkpoints. Fixed-Dependency-Interval (FDI) [8, 12] and Fixed-Dependency-After-Send (FDAS) [12] maintain and propagate dependency vectors. They force the dependency vector of a process to remain unchanged during an entire checkpoint interval (FDI) or after the first message-send event of an interval (FDAS).

The RDT protocol proposed by Baldoni, Helary, Mostefaoui, and Raynal (BHMR) was the first protocol to consider visibly doubled Z-paths [1]. Afterwards, Baldoni, Helary, and Raynal have presented a family of RDT protocols, including a refined version of BHMR [3]. Tsai, Kuo, and Wang have proved that BHMR never takes more forced checkpoints than FDAS [11]. However, the more elaborated condition used by BHMR requires the propagation of an $O(n^2)$ matrix of booleans, as described in Section 4.

Recently, we have proposed an RDT protocol, called RDT-partner, that breaks only non-trivially doubled PMM-paths [6]. RDT-partner requires only $O(n)$ control information
and our simulation results have shown that it takes virtually the same number of forced checkpoints as BHMR. However, given the results presented in this article, it is now also possible to break all non-visibly doubled PMM-paths in $O(n)$.

References


