A graph transformation-based approach for the validation of checkpointing algorithms in distributed systems

Houda Khlif, Hatem Hadj Kacem
ReDCAD Laboratory
FSEGS, University of Sfax
Sfax, Tunisia
houdakhlif@gmail.com
Hatem.Hadjkacem@fsegs.rnu.tn

Saúl E. Pomares Hernandez [1,2,3], Cédric Eichler[2,3]
[1]Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE)
Luis Enrique Erro 1, C.P. 72840, Tonantzintla, Puebla, Mexico
[2]CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
[3]Univ de Toulouse, LAAS, F-31400 Toulouse, France
spomares@inaoep.mx
ceichler@laas.fr

Ahmed Hadj Kacem
ReDCAD Laboratory
FSEGS, University of Sfax
Sfax, Tunisia
Ahmed.Hadjkacem@fsegs.rnu.tn

Alberto Calixto Simón
Universidad del Papaloapan, UNPA
Av, Ferrocarril s/n, C.P. 68400, Loma Bonita, Oaxaca, Mexico
acalixto@unpa.edu.mx

Abstract*—Autonomic Computing Systems are oriented to prevente the human intervention and to enable distributed systems to manage themselves. One of their challenges is the efficient monitoring at runtime oriented to collect information from which the system can automatically repair itself in case of failure.** ***Quasi-Synchronous Checkpointing **is a well-known technique, which allows processes to recover in spite of failures. Based on this technique, several checkpointing algorithms have been developed. According to the checkpoint properties detected and ensured, they are classified into: Strictly Z-Path Free (SZPF), Z-Path Free (ZPF) and Z-Cycle Free (ZCF). In the literature, the simulation has been the method adopted for the performance evaluation of checkpointing algorithms. However, few works have been designed to validate their correctness. In this paper, we propose a validation approach based on graph transformation oriented to automatically detect the previous mentioned checkpointing properties. To achieve this, we take the vector clocks resulting from the algorithm execution, and we model it into a causal graph. Then, we design and use transformation rules oriented to verify if in such a causal graph, the algorithm is exempt from non desirable patterns, such as Z-paths or Z-cycles, according to the case.**

I. INTRODUCTION

As computing systems have reached a level of complexity, their management has become increasingly difficult. As a result, the initiative of autonomic computing has been introduced to prevent human intervention and enable the system to manage itself. An Autonomic Distributed System is considered as a set of geographically distributed autonomic components that communicate/collaborate among them. In general, the autonomic computing has four elements: self-configuration, self-healing, self-optimizing and self-protecting. One open challenge in self-healing is the efficient monitoring at runtime oriented towards the collection of information, from which the system itself will detect problems that result from failures in software and/or hardware components, diagnose and initiate a corrective action without distributing system applications. Checkpointing is a well-known technique that allows processes to make progress in spite of failures [13]. Its basic idea is to start the system from a consistent state when failures occur. Indeed, when global states are periodically recorded during the system execution, they are called global checkpoints. Thus, a global checkpoint is a set of local checkpoints, where each one presents the recorded states of a process. A global checkpoint is consistent if no local checkpoint occurs before another, that is, there is no causal path from one checkpoint to another, in the sense that a message (or a sequence of messages) sent after one checkpoint is received before the other [7]. Checkpointing algorithms are organized into three classes: asynchronous, synchronous and quasi-synchronous [9]. In asynchronous checkpointing, also known as uncoordinated checkpointing, each process takes its checkpoints independently, which leads to the domino effect. The domino effect exists due to a dependency relation between checkpoints called zigzag path or Z-path [11]. In synchronous checkpointing, or coordinated checkpointing, inorder to avoid the domino effect, processes coordinate their checkpoints by the addition of control messages to form consistent states. In quasi-synchronous checkpointing, or Communication Induced Checkpointing (CIC), coordination is achieved by piggybacking control information on application messages and taking forced local checkpoints in case of dangerous patterns, such as a Z-path, in order to avoid the domino effect.

Several algorithms, which propose different methods to force checkpoints and produce checkpoint and communication patterns with different properties, have been developed. They are classified into: Strictly Z-Path Free (SZPF), Z-Path Free (ZPF) and Z-Cycle Free (ZCF). However, few works have

dealt with the correctness of such algorithms. In this paper, we propose a validation approach for checkpointing algorithms. In fact, to validate the correctness of a checkpointing algorithm, we propose modelling its execution by a graph. Then, we use graph transformation approaches to verify if in such a graph, the algorithm is exempt from dangerous patterns like Z-paths and Z-cycles.

This paper is structured as follows. Section 2 defines the system model and background, describes the classification of quasi-synchronous checkpointing, and defines the principle approaches of graph transformation. Section 3 shows some related works. In section 4, we describe the process of our approach, and we present a set of transformation rules that we have designed. Finally, we summarize our contributions and suggest new research.

II. PRELIMINARIES

A. System Model

Processes. The system under consideration is composed of a set of processes $P = \{p_1, p_2, \cdots, p_n\}$. The processes present an asynchronous execution and communicate only by message passing.
Messages. We consider a finite set of messages M, where each message $m \in M$ is sent considering an asynchronous reliable network that is characterized by no transmission time boundaries, no order delivery, and no loss of messages. The set of destinations of a message m is identified by $Dest(m)$.
Events. We consider two types of events: internal and external events. An internal event is a unique action that occurs at a process p in a local manner and which changes only the local process state. We denote the finite set of internal events as R. The set R represents the set of relevant events. We consider only the checkpoints as internal events. We denote by C_i^x the x^{th} checkpoint of process p_i. The sequence of events occurring at p_i, between C_i^{x-1} and C_i^x $(i > 0)$, is called a checkpoint interval (denoted as I_i^x). On the other hand, while an external event is also a unique action that occurs at a process, it is seen by other processes, thus, affecting the global state of the system. The external events considered in this paper are the send and delivery events. Let m be a message. We denote by $send(m)$ the emission event and by $delivery(p, m)$ the delivery event of m to participant $p \in P$. The set of events associated to M is the set $E(M) =$
$\{send(m) : m \in M\} \cup \{delivery(p, m) : m \in M \wedge p \in P\}$.
The whole set of events in the system is the finite set $E = R \cup E(M)$. The distributed computation is modeled by the partially ordered set $\hat{E} = (\hat{E}, \rightarrow)$, where $\rightarrow$ denotes Lamports Happened Before Relation [7].

B. Background and Definitions

Happened Before Relation (HBR). The definition of HBR is the following:

Definition 1. *The Happened Before Relation (HBR) [7], "$\rightarrow$", is the smallest relation on a set of events E satisfying the following conditions:*

1) *If a and b are events belonging to the same process, and a was originated before b, then $a \rightarrow b$.*
2) *If a is the sending of a message by one process, and b is the receipt of the same message in another process, then $a \rightarrow b$.*
3) *If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$.*

The Happened Before Relation among the set of relevant events $R \in E$, is defined as follows:

- If C_i^x and C_j^y are checkpoint (relevant) events belonging to the same process, and C_i^x was originated before C_j^y, then $C_i^x \rightarrow C_j^y$.
- If C_i^x is the sending of a message by one process, and C_j^y is the receipt of the same message in another process, then $C_i^x \rightarrow C_j^y$.
- If $C_i^x \rightarrow C_j^y$ and $C_j^y \rightarrow C_k^z$, then $C_i^x \rightarrow C_k^z$.

Z-path and Z-cycle. Netzer et al. [10] defined the notion of zigzag path (z-path) as a genaralization of HBR, as follows:

Definition 2. *Z-path exists from C_i^p to another C_j^q iff there are messages $m_1, m_2, ..., m_l$ such that:*

1) *m_1 is sent by process p after C_i^p,*
2) *if m_k $(1 \leq k < l)$ is received by process r, then m_{k+1} is sent by r in the same or at a later checkpoint interval (although m_{k+1} may be sent before or after m_k is received), and*
3) *m_l is received by process q before C_j^q.*

Helary et al defined the following in [4].

Definition 3. *A Z-path $[m_1, ..., m_q]$ is causal, iff for each pair of consecutive messages m_α and $m_{\alpha+1}$:*
delivery(m_α) $\rightarrow$ send($m_{\alpha+1}$). Otherwise, it is a non causal Z-path.

Definition 4. *A local checkpoint C_j^y Z-depends on a local checkpoint C_i^x , $C_i^x \xrightarrow{Z} C_j^y$, if:*

1) *$j = i$ and $y > x$, or*
2) *there is a Z-path from C_i^x to C_j^y.*

Definition 5. *A Z-cycle is a Z-dependency from a local checkpoint C_i^x to itself: $C_i^x \xrightarrow{Z} C_i^x$.*

Theorem 1. *The length of a Z-cycle (or Z-path) is l if the Z-cycle (or Z-path) is formed by l messages m_1, m_2, ..., m_l.*

Definition 6. *A communication and checkpoint pattern (CCP) is a pair ($\hat{E}$, $R_{\hat{E}}$) where $\hat{E}$ is a partially ordered set modeling a distributed computation, and $R_{\hat{E}}$ is a set of local checkpoints defined on $\hat{E}$.*

An example of Communication and checkpoint pattern is given in Figure 1. In this example, the messages [m_1,m_2] form a Z-cycle of length two involving C_2^2, and [m_5,m_4,m_3] form a Z-cycle of length three involving C_3^3.

Theorem 2. *The following properties of a communication and checkpoint pattern ($\hat{E}$,R) are equivalent:*

1) *($\hat{E}$,R) has no Z-cycle.*
2) *It is possible to timestamp its local checkpoints in such a manner that*

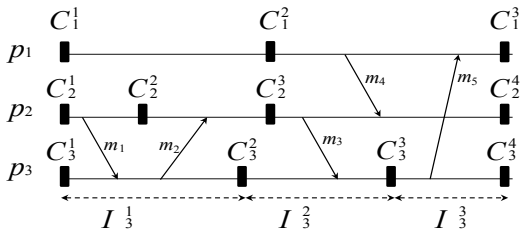


Fig. 1. A communication and checkpoint pattern

$$A \xrightarrow{Z} B \Rightarrow A.t < B.t$$

where t is a logical clock as defined by Lamport [7].

C. Classification of quasi-synchronous checkpointing

A local checkpoint that cannot be part of a consistent global snapshot is said to be useless. A local checkpoint that can be part of a consistent global snapshot is called a useful checkpoint. The main advantage of a quasi-synchronous checkpointing algorithm is that it can reduce the number of useless checkpoints. Quasi-synchronous checkpointing algorithms are classified into three different classes, namely, Strictly Z-Path Free (SZPF), Z-Path Free (ZPF), and Z-Cycle Free (ZCF) [9].

Strictly Z-path free checkpointing eliminates all the noncausal Z-paths between checkpoints altogether.

Definition 7. *A checkpointing pattern is said to be strictly Z-path free (or SZPF) if there exists no noncausal Z-path between any two (not necessarily distinct) checkpoints.*

In a ZPF system, it is possible to prevent useless checkpoints by eliminating only those noncausal Z-paths in which there is no sibling causal path. The ZPF model is defined below:

Definition 8. *A checkpointing pattern is said to be Z-path free (or ZPF) iff for any two checkpoints A and B, a Z-path exists from A to B iff there exists a causal path from A to B.*

In a ZCF model, only Z-cycles are prevented. A ZCF model is defined as follows:

Definition 9. *A checkpointing pattern is said to be Z-cycle free (or ZCF) iff none of the checkpoints lie on a Z-cycle.*

All checkpoints taken in SZPF, ZPF and ZCF systems are useful. The construction of consistent global checkpoints is more difficult in a ZCF than the other systems because of the existence of noncausal Z-paths. However, a ZCF system has the potential to have the lowest checkpointing overhead as it takes forced checkpoints only to prevent Z-cycles. Figure 2 summarizes the relationship between the various quasi synchronous checkpointing models. As shown in this Figure, a SZPF system is a ZPF system, but the converse is not true. Likewise, a ZPF system is a ZCF system, but the converse is not true.

D. Graph Transformation

Graph Transformation is a rule-based approach targeted towards the modifications on a graph [14], [2], [1]. A Graph

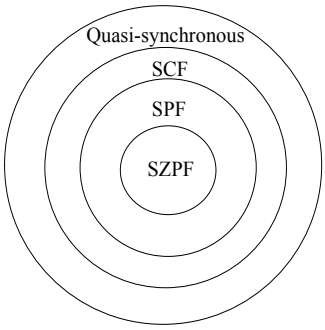


Fig. 2. Relationship between the various quasi synchronous checkpointing models

Transformation Rule is described by a pair of graphs $r = (L, R)$, where L is called the left-hand side graph and R is called the right-hand side graph. Applying the rule $r = (L, R)$ means finding a match of L in the source graph and replacing L by R. The suppression of the occurrence of L in G may cause the appearance of edges without a starting node or a terminating node or both. Those edges are called "dangling edges". Two principle approaches have dealt with this problem, which are the SPO approach and the DPO approach [1].

The Single PushOut Approach (SPO). A rule of type SPO is a production of the form (L, R). Its application to a graph G is related to the existence of an occurrence of L in G. The application of the SPO rule to a graph G involves the removal of the graph corresponding to $Del = (L \backslash (L \cap R))$ and the addition of the graph corresponding to $Add = (R \backslash (L \cap R))$.

The Double PushOut Approach (DPO). A rule of type DPO is a production of the form (L, K, R), where K is used to specify clearly the part to preserve after applying the rule. If both conditions of existence of the occurrence of L and absence of suspended edges are checked, the application of the rule involves the removal of the graph corresponding to the occurrence of $Del = (L \backslash K)$ and the addition of a copy of the graph $Add = (R \backslash K)$.

Neighborhood Controlled Embedding Approach (NCE). The NCE mechanism [14] is based on the specification of the so-called connection instructions. These instructions are based on the labels of nodes to define new edges to be introduced. The connection instructions are described by a pair (n, δ). The execution of this instruction connection involves the introduction of an edge between the added node n and all the neighboring nodes of the removed node, whose label is δ. dNCE (d for edge label), eNCE (e for edge label), and edNCE are extensions of NCE approach. The dNCE connection instructions are described by a triplet (n, δ, d), where $d \in \{in, out\}$ is for controlling the direction of the edges. The eNCE connection instructions are described by a triplet $(n, p/q, \delta)$, where p and q are edge labels. The edNCE is the combination of the eNCE and dNCE approaches. The edNCE connection instructions are of the form $(n; p/q; \delta; d; d')$. The execution of this instruction implies the introduction of an edge in the direction indicated by d' between the node n and all nodes n' that are p-neighbours and d-neighbours (in-neighbours if d=in and out-neighbours otherwise) of the removed node.

III. Related Work

Finding a method to construct consistent global checkpoints in a ZCF system has been an open problem. The impossibility of designing an optimal ZCF quasi-synchronous checkpointing algorithm has been treated by Tsai et al [17]. In the last few years, some ZCF quasi-synchronous checkpointing algorithms have been developed. For example, the Fully Informed (FI) algorithm of Helary et al [4], the Fully Informed aNd Efficient (FINE) algorithm of Luo et al. [8], the Delayed Communication-Induced Checkpointing (DCFI) algorithm [15] and The Scalable Fully Informed (SF-I) algorithm [16] of Calixto et al. According to our knowledge, the present work introduces the first solution based on graph transformation for the validation of checkpointing properties that can be used with any CIC algorithm.
Wang [18], [19] has defined a graph called "Rollback depedency graph" to show Z-paths in a distributed computation. It is easy to detect Z-paths in such a graph, but detecting Z-cycles is a critical problem. Taesoon Park and Heon Y. Yeom [12] have proposed a scheme for detecting Z-cycles of length two. Such scheme takes forced checkpoints to break them. Then, Chin-Lin Kuo and Yuo-Ming Yeh [6] have proposed an inline distributed algorithm to detect all Z-cycles of long length and their involved checkpoints. This algorithm conceptualizes an appropriate data structure to express Z-path and Z-cycle. It requires much piggybacked Z-path information, but it detects, for a distributed computation, all the existing Z-cycles and their involving checkpoints. In order to use this last solution for validation purposes, such algorithm must be executed at runtime in a simultaneous way along with the checkpointing algorithms, which implies an expensive and additional use of memory, processing and bandwidth resources.

IV. Approach

Figure 3 ullistrates the general process of our approach. We have choosen to use the Graph Matching and Transformation Engine (GMTE)[1] as it is able to search small and medium graph patterns in huge graphs in a short time. It receives as input graph descriptions and transformation rules written in XML [3]. In our approach, the GMTE receives a checkpoint graph that models the execution of a checkpointing algorithm, for which it applies a transformation rule presenting dangerous patterns to show if in such a graph the algorithm is exempt from these patterns. Our solution is based on two main approaches which are SPO approach and edNCE approach. The SPO approach is to deal with suspended egdes as it provides power to remove this kind of edges greater than the DPO approach. The edNCE approach is for the addition and the removal of nodes using the connection instructions.

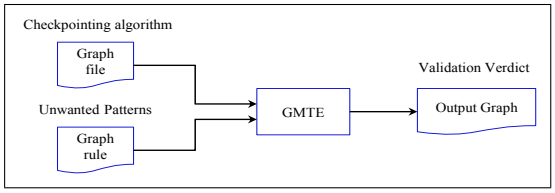


Fig. 3. Validation process

[1]GMTE is available at http://homepages.laas.fr/khalil/GMTE/

A graph transformation rule applied to an HBR graph. The HBR graph presents the causal relations of the system. The graph presented in Figure 4 is the HBR graph corrsponding to the scenario of Figure 1. As shown in this Figure, a causal graph contains three types of edges:

- A local relation that connects two successive checkpoints C_i^x and C_i^{x+1}, belong to the same process p_i, is denoted by the label "c".
- A direct relation between two checkpoints C_i^x of p_i and C_j^y of p_j is denoted by the label "d".
- A transitive relation is denoted by the label "t".

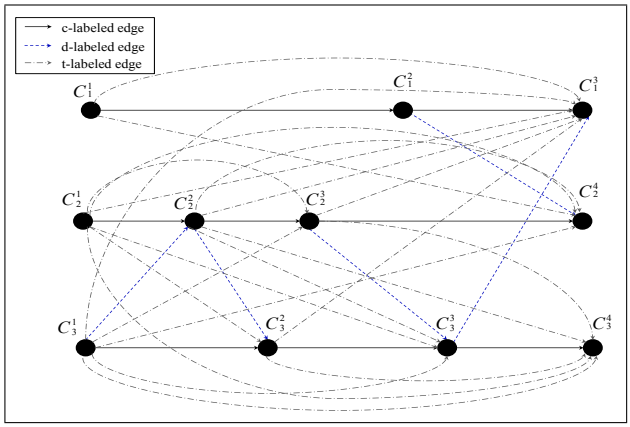


Fig. 4. Example of an HBR graph

Figure 5 presents the general patterns of Z-paths and Z-cycles. A Z-path, in an HBR graph, is in the form of ($n_1 \xrightarrow{d} n_3$, $n_2 \xrightarrow{c} n_3$, $n_3 \xrightarrow{d} n_4$). It can be either of length two or more. A Z-cycle, in an HBR graph, is in the form of ($n_1 \xrightarrow{d} n_2$, $n_2 \xrightarrow{c} n_3$, $n_3 \xrightarrow{d} n_1$). It can also be of length two or more.

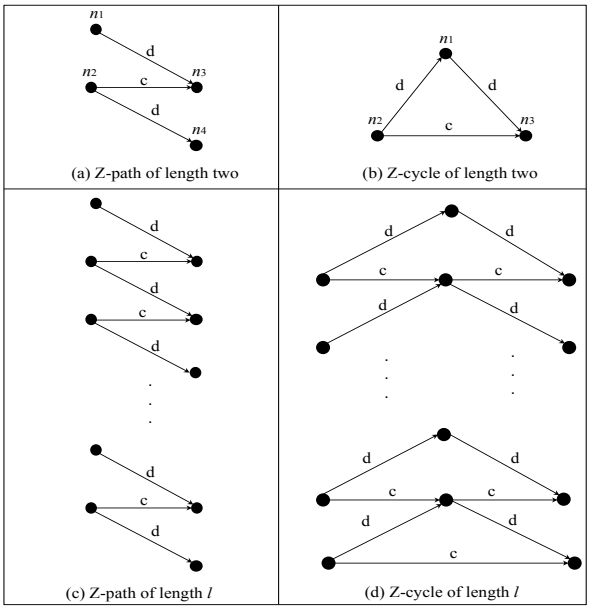


Fig. 5. General patterns of Z-paths and Z-cycles in an HBR graph

We start by eliminating all t-labeled edges, as they can be involved neither in a Z-path nor in a Z-cycle. For this, we use the rule r_1. This rule is of type SPO. Its application implies the removal of all t-labeled edges. An example of the application of rule r_1 to an HBR graph is given in Figure 6.

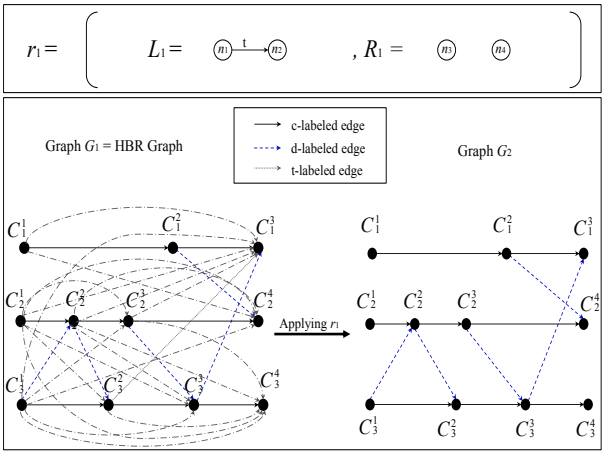


Fig. 6. The application of rule r_1

In the first time, we verify if the output graph of r_1 is exempt from Z-paths. For this, we implement the SPO rule r_2 that we present in Figure 7. Its application implies the addition of a z-labeled edge between the two checkpoints (nodes) involved in a Z-path. An illustrative example is given in Figure 7. The application of r_2 to graph G_2 shows the existence of two Z-paths: the first is from C_1^2 to C_3^3 ($C_1^2 \xrightarrow{d} C_2^4$, $C_2^3 \xrightarrow{c} C_2^4$, $C_2^3 \xrightarrow{d} C_3^3$) and the second is from C_3^3 to C_2^4 ($C_3^3 \xrightarrow{d} C_1^3$, $C_1^2 \xrightarrow{c} C_1^3$, $C_1^2 \xrightarrow{d} C_2^4$).

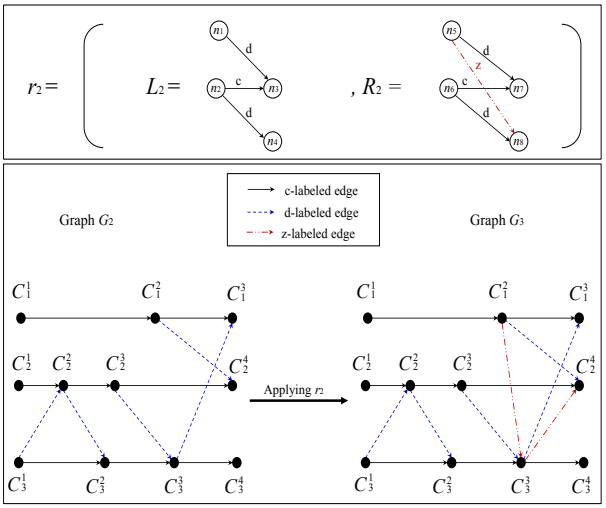


Fig. 7. The application of rule r_2

The occurence of the label "z"in the resulting graph implies that the system is not ZPF. So, it can also contain Z-cycles. Then, we implement the rules r_3, r_4 and r_5 to verify such property. These rules are of type SPO, and they are sequentially executed.

Rule r_3, presented in Figure 8, is to detect a long Z-path. It is applied to G_3, the output graph of r_1. Two successive Z-paths form another Z-path ($n_1 \xrightarrow{z} n_3$, $n_2 \xrightarrow{d} n_3$, $n_3 \xrightarrow{z} n_4$). We denote this model by adding a z-labeled edge between n_1 and n_4. In the HBR graph that we have previously used, there is no case of long Z-path. G_4, the output graph of r_3, is equivalent to graph G_3. For this, we illustrate the result of the application of rule r_3 in the general case (see Figure 8).

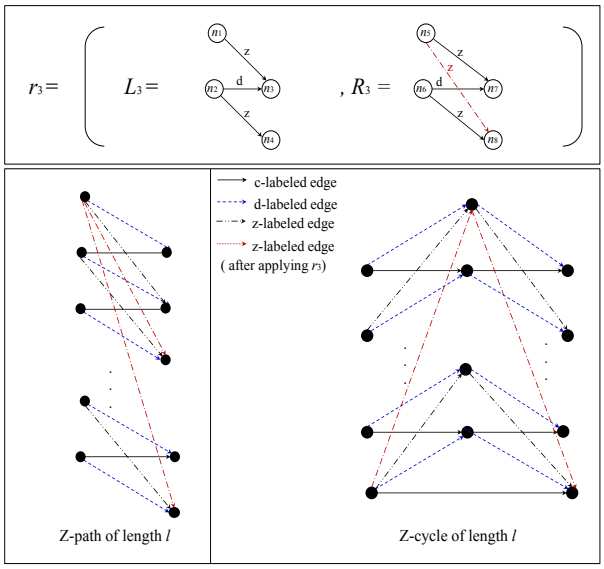


Fig. 8. The application of rule r_3

Rule r_4 (presented in Figure 9) is used to detect Z-cycles of length two. Its application implies replacing the node involved in such a Z-cycle by a zc-labeled node. In this level, it is necessary to use the connection instructions to link the added node and all nodes that are d-neighbours (in-neighbours if d=in and out-neighbours otherwise) of the removed node.This rule is applied to graph G_4. Figure 9 illustrates the application of r_4. In this example, only one Z-cycle of length two exists involving C_2^2 ($C_3^1 \xrightarrow{d} C_2^2$, $C_3^1 \xrightarrow{c} C_3^2$, $C_3^2 \xrightarrow{d} C_2^2$).

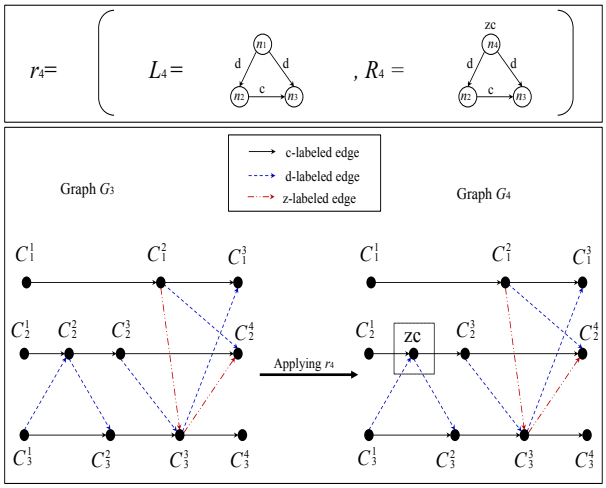


Fig. 9. The application of rule r_4

Finally, we apply rule r_5 to G_5 to detect long Z-cycles. As a result, a node involved in a long Z-cycle is removed and replaced by a new zc-labeled node. In this level, it is also necessary to use the connection instruction because of the removal and the addition of nodes. Figure 10 presents rule r_5 and illustrates its application. In the given example, a z-cycle exists involving C_3^3 ($C_1^2 \xrightarrow{z} C_3^3$, $C_1^2 \xrightarrow{d} C_2^4$, $C_3^3 \xrightarrow{z} C_2^4$).

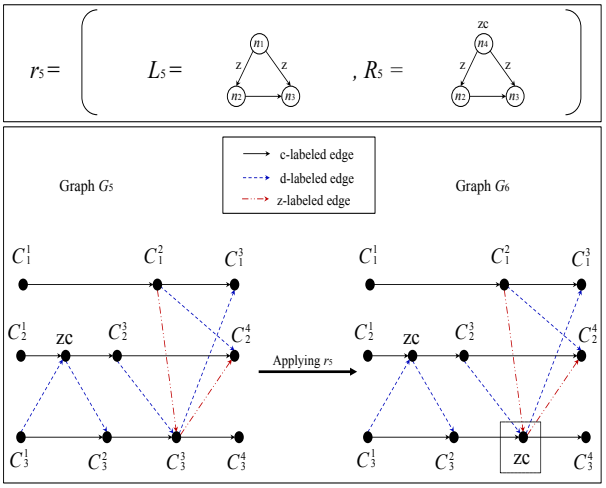


Fig. 10. The application of rule r_5

By applying the set of transformation rules previously presented to a causal graph, we can detect all existing Z-paths and Z-cycles, whether they are of length two or more. Then, we can decide for any checkpointing algorithm if it is ZPF on the one hand, and if it is ZCF on the other hand. The use of graph transformation approaches ensures correct results without much need of time and space. As a result, we obtain an efficient validation approach, which verifies the correctness of checkpointing algorithms in a less costly way.

V. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a validation approach for checkpointing algorithms in distributed systems. We have used graph transformation approaches to validate the correctness of such algorithms in a less costly way. Firstly, we have designed a set of transformation rules to verify in a causal graph, the existence of non desirable patterns like Z-paths and Z-cycles. The application of these rules shows all existing Z-paths and Z-cycles in such a graph regardless of its length. The cost reduction is explained by the fact that the designed rules are not executed at runtime. With the obtained results, we can validate if the algorithm is ZPF in the first time and if it is ZCF in the second.

As an on-going work, we aim to design transformation rules oriented to the Minimal Causal and Set Representation (MCSR)[2] tool that constructs, for a distributed computation, the minimal causal graph (IDR graph) at a single event level. For this, the MCSR uses the Immediate Dependency Relation (IDR) [5]. The IDR graph greatly reduces the state-space of a system. This is the reason why finding an efficient method to detect non desirable patterns in such graphs can be an important contribution in term of cost reduction.

[2]MCSR is available at http://homepages.laas.fr/khalil/GMTE/

REFERENCES

[1] H. Ehrig, H.J. Kreowski, and G. Rozenberg. Tutorial introduction to the algebraic approach of graph grammars based on double and single pushouts. In *Graph-Grammars and Their Application to Computer Science*, volume 532 of *LNCS*. Springer, March 5-9 1990.

[2] J. Engelfriet and G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformation*, chapter Node Replacement Graph Grammars, pages 1–94. World Scientific Publishing, 1997.

[3] M. A. Hannachi, I. B. Rodriguez, K. Drira, and S. E. Pomares-Hernandez. GMTE: A tool for graph transformation and exact/inexact graph matching. In *Graph-Based Representations in Pattern Recognition. 9th IAPR-TC-15 International Workshop, GbRPR 2013, Vienna, Austria*, volume 7877 of *LNCS*. Springer, 2013.

[4] J. M. Helary, A. Mostefaoui, R. H. B. Netzer, and M. Raynal. Communication-based prevention of useless checkpoints in distributed computations. *Distributed Computing*, 13:29–43, January 2000.

[5] S.E. Pomares Hernandez, J. Fanchon, and K. Drira. *The Immediate Dependency Relation: An Optimal Way to Ensure Causal Group Communication*. Singapore University Press and World Scientific Publications, 2004.

[6] C. L. Kuo and Y. M. Yeh. An algorithm for detecting z-cycles in distributed computing system. In *Int. Computer Symposium*, pages 1124–1133. Int. Computer Symposium, December 2004.

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

[8] Y. Luo and D. Manivannan. Fine: A fully informed and efficient communication-induced checkpointing protocol for distributed systems. *Journal of Parallel and Distributed Computing*, 69:153–167, February 2009.

[9] D. Manivannan and M. Singhal. Quasi-synchronous checkpointing: Models, characterization, and classification. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):703–713, 1999.

[10] R. H. B. Netzer and J. Xu. Necessary and sufficient conditions for consistent global snapshots. *IEEE*, 6:165–169, February 1995.

[11] Robert H. B. Netzer and Jian Xu. Adaptive message logging for incremental replay of message-passing programs. Technical report, Brown University, Providence, RI, USA, 1993.

[12] T. Park and H. Y. Yeom. Application controlled checkpointing coordination for fault-tolerant distributed computing systems. *Parallel Computing*, 26:467–482, 2000.

[13] B. Randell, P. A. Lee, and P. C. Treleaven. Reliability issues in computing system design. *ACM*, 10(2):123–166, June 1978.

[14] G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 3. World Scientific Publishing, 1997.

[15] A. C. Simon, S. E. Pomares Hernandez, and J. R. Pilar Cruz. A delayed checkpoint approach for communication-induced checkpointing in autonomic computing. In *Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, June 2013.

[16] A. C. Simon, S. E. Pomares Hernandez, J. R. Pilar Cruz, P. Gomez-Gil, and K. Drira. A scalable communication-induced checkpointing algorithm for distributed systems. *IEICE TRANSACTIONS on Information and Systems*, E96-D(4):886–896, April 2013.

[17] J. Tsai, Y. Wang, and S. Kuo. Evaluations of domino-free communication-induced checkpointing protocols. *Information Processing Letters*, 69:1–69, 1998.

[18] Y. M. Wang. Maximum and minimum consistent global checkpoints and their applications. In *Symposium on Reliable Distributed Systems*, pages 86–95, 1995.

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