A Mechanism for the Causal Ordered Set Representation in Large-Scale Distributed Systems

Houda Khlif, Hatem Hadj Kacem, Saúl Eduardo Pomares Hernández, Ahmed Hadj Kacem

A mechanism for the Causal Ordered Set Representation in Large-Scale Distributed Systems

Houda Khlif, Hatem Hadj Kacem
ReDCAD Laboratory
University of Sfax, Tunisia

Saúl E. Pomares Hernandez
INAOE, Tonantzintla, Mexico
CNRS, LAAS, F-31400 Toulouse, France

Ahmed Hadj Kacem
ReDCAD Laboratory
University of Sfax, Tunisia

***Abstract*—Distributed systems have undergone a very fast evolution these last years. Large-scale distributed systems have become an integral part of everyday life with the development of new large-scale applications, consisting of thousands of computers and supporting millions of users. Examples include global Internet services, cloud computing systems, "big data" analytics platforms, peer-to-peer systems, wireless sensor networks and so on. The recent research addresses questions related to the way one may design, build, operate and maintain large-scale distributed systems. An other question related to such area, is how to represent causal dependencies in such systems in a minimal way. In general, causal dependencies can be established according to the Happened-Before Relation (HBR), which was introduced by Lamport. The HBR is a strict partial order, and therefore, one main problem linked to it is the combinatorial state explosion. The Immediate Dependency Relation (IDR) and the Causal Order Set Abstraction (CAOS) present a solution for such a problem. In this paper, we propose a mechanism which uses the concepts HBR, IDR, CAOS to model a large-scale distributed system execution in the form of the minimal graph (IDR graph) and the compact graph (CAOS graph). This mechanism is implemented in C++. The results of its execution are given here. The resultant causal graphs can be used for different purposes, such as for the design of more efficient algorithms, validation, verification, and/or the debugging of the existing ones, among others.**

***Keywords*—Distributed system; Happened Before Relation; Immediate Dependency Relation; Causal Ordered Set Abstraction**

I. Introduction

In the last few decades, distributed systems have become an essential part of our lives. Today, many of the most important applications are distributed and new large-scale applications have emerged such as digital library systems, electronic commerce environement, office automation workflow and collaboration envirenment. Consequently, new requirement have emerged for large-scale distributed systems such as solutions for efficient design, synchronization, self management, fault tolerance, rollback recovery and so on. Such solutions are based on causal ordering property. The use of causal ordering provides built-in message synchronization, reduces the nondeterminism in a distributed computation, and provides an equivalent of the FIFO property at a party communication level. Causal ordering guarantees that actions, such as requests or questions, are received (viewed) before their corresponding reactions, results, or responses. In general, causal dependencies can be established according to the Happened-Before Relation (HBR), which was introduced by Lamport [3]. The HBR is a strict partial order (transitive, irreflexive and antisymmetric), and therefore, one main problem linked to it is the combinatorial state explosion [1]. To deal with this problem, Hernandez et al. [8] define the Immediate Dependency Relation (IDR) as the transitive reduction of the HBR. The IDR is the unique minimal expression of the HBR. Based on this relation, the Causal Ordered Set Abstraction (CAOS) [7] was introduced to pass from a single event level to an event set level. The CAOS allows a minimal set representation of causal dependencies. In this paper, we propose a mechanism which is based on the CAOS to model a large-scale distributed system execution in a minimal way. It has as input a database of vector clocks and as output three causal graphs: the fully-causal HBR original graph, the IDR graph, and the CAOS graph. To do this, three basic algorithms are proposed and implemented in c++: VC2HBR for the generation of the HBR graph, HBR2IDR for the generation of the IDR graph, and finally IDR2CAOS for the generation of the CAOS graph. The CAOS graph drastically reduces the state-space of a system. The resultant causal graphs can be used for different purposes, such as for the design of more efficient algorithms, validation, verification, and/or the debugging of the existing ones, among others. In this paper, we present a case study that shows the usefulness of the causal graphs for validation purposes in checkpointing protocols.

The rest of this paper is structured as follows. Section II presents the system model and associated definitions. The proposed mechanism is presented in Section III. A case study is presented in section IV to illustrate how the causal graphs can be used for validation purposes. Conclusion and future work are finally discussed in Section V.

II. Background and Definitions

A. System Model

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. M is a finite set of messages, where each message $m \in M$ is sent considering an asynchronous reliable network which 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 $Dest(m)$.

Two types of events are considered here: internal and external events. An internal event is a unique action which 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 I. An external event is also a unique action which occurs at a process, it is seen by other processes, and thus, affects the global state of the system. The external events considered in this paper are the send and delivery events. Let $m \in M$

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: $E(M) = send(m) \cup delivery(p, m)$. The whole set of events in the system is the finite set $E = I \cup E(M)$. Each event $e \in E$ is identified by a tuple $id(e) = (p, x)$, where $p \in P$ is the producer of e, and x is the local logical clock for events of p, when e is carried out. When we need to refer to a specific event we use the notation $e_{p,x}$.

B. Happened-Before Relation (HBR)

The Happened Before Relation for single events was defined by Lamport [3]. This relation establishes causal precedence dependencies over a set of events. The HBR is a strict partial order (transitive, irreflexive and antisymmetric). It is defined as follows:

Definition 1. *The causal relation "$\rightarrow$" is the smallest relation on a set of events E satisfying the following conditions:*

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

By using Definition 1, Lamport defines that a pair of events is concurrently related "$a||b$"if the following condition is satisfied:

$$a||b \text{ if } \neg(a \rightarrow b \vee b \rightarrow a)$$

The HBR for sets is also a strict partial order. It is formally defined as follows:

Definition 2. *The causal relation "$\rightarrow$" is established at the set level by satisfying the following conditions:*

- *$A \rightarrow B$ if $a \rightarrow b$, $\forall(a, b) \in A \times B$,*
- *$A \rightarrow B$ if $\exists C \mid (A \rightarrow C \wedge C \rightarrow B)$.*

However, according to the specification of ordered sets presented by Shimamura et al. [10] and Hernandez et al [9], which assume a local total ordering among the events which compose a set, the causal relation for sets can be accomplished only in terms of the endpoints as follows:

Property 1. *The relation "$\rightarrow$" is accomplished at the ordered set level if the following conditions are satisfied:*

- *$A \rightarrow B$ if $a^+ \rightarrow b^-$*
- *$A \rightarrow B$ if $\exists C \mid (a^+ \rightarrow c^- \wedge c^+ \rightarrow b^-)$*

where a^+ and b^- are the right and the left endpoints of A and B, respectively, c^- and c^+ are the endpoints of C.

The concurrent relation for ordered sets is defined as follows:

Definition 3. *Two ordered sets of events, A and B, are said to be concurrently related "A||B" if the following condition is satisfied:*

$$A \mid\mid B \text{ if } a \mid\mid b, \forall a \in A, \forall b \in B$$

C. Vector Clocks

The Vector Clocks' algorithm was simultaneously developed by Fidge [2] and Mattern [4]. It was created to detect the causal relations among events in a distributed system. A Vector Clock is an array of *logical clocks* of size n, where n is the number of processes in a system. A Vector Clock captures the causal state of the system. For each event e generated, a Vector Clock is associated, and it is denoted by $VC(e)$. In general, the Vector Clocks' algorithm is defined as follows:

- Each process p_i is equipped with a Vector Clock VC_i. A process p_i increments its own Vector Clock for each event e (internal or external) in the form:
$$VC_i[i] := VC_i[i] + 1$$
- In each message m sent by a process p_i, the current Vector Clock VC_i is piggybacked in m. At the reception of a message m by a process p_j, it updates its clock as follows:
$$VC_j := max(VC_j, VC_i)$$
Where max gets the maximum values of each pair of elements in the Vector Clocks.

Based on the Vector Clocks' algorithm, the causal dependencies can be established as follows:

Definition 4. *For a pair of events $a, b \in E$, the event a causally precedes the event b if the following condition is satisfied:*

$$a \rightarrow b \text{ if } VC_i(a) < VC_j(b)$$

D. Immediate Dependency Relation (IDR)

The IDR is the transitive reduction of the HBR [8]. It is denoted by "$\downarrow$", and it is defined as follows:

Definition 5. *Two events $a, b \in E$ have an immediate dependency relation "$\downarrow$" if the following restriction is satisfied:*

$$a \downarrow b \text{ if } a \rightarrow b \text{ and } \forall c \in E, \neg(a \rightarrow c \rightarrow b)$$

Thus, an event a causal-immediately precedes an event b, if and only if no other event $c \in E$ exists, such that c belongs to the causal future of a and to the causal past of b.

Property 2. *For all pair of events $a, b \in E$, $a \neq b$:*

if $\exists c \in E$ such that ($a \downarrow c$ and $b \downarrow c$) or ($c \downarrow a$ and $c \downarrow b$), then $a \mid\mid b$

E. Causal Ordered Set Abstraction (CAOS)

Assuming the poset E = ($\hat{E}$,$\rightarrow$) as the model adopted for a distributed computation, the objective of CAOS is to establish over this poset the rules of association of events and the conditions of ordering between the resulting sets [7]. We consider a finite collection S of ordered sets of events, where each set $W \in S$ is a set of events $W \subseteq E$. The elements of a set are ordered according to the IDR. Such elements compose a causal path (linearization) from an event e_1 to an event e_k such that $W = \{e_1 \downarrow e_2 \downarrow \cdots \downarrow e_k\}$. We denote by w^- and w^+ the endpoint events of $W(w^- = e_1$ and $w^+ = e_k)$. The definition of CAOS is made up of three parts:

TABLE I. CAOS SPECIFICATION

1. A new set $W(e)$ is created in S when:	
C1. $\exists e \in E, \neg(\exists Z \in N : e \in Z)$	1
$R_1 = \{e : \emptyset \downarrow e\}$ or	2
$W(e) \leftarrow R_2 = \{e : \exists (e_a, e_b) \in E; e_a \downarrow e \wedge e_a \downarrow e_b\}$ or	3
$R_3 = \{e : \exists (e_a, e_b) \in E; e_a \downarrow e \wedge e_b \downarrow e\}$	4
2. The rest of the events $e' \in E$ are assigned to a set $W(e) \in N$ as follows:	
C2. $\exists W(e) \in N, \exists e' \in E, \neg(\exists Z \in N : e' \in Z)$	5
$W(e) \leftarrow R_4$.	6
$W(e) \cup \{e' : \exists e_a \in W(e), \exists e_b \in E; e_a \downarrow e' \wedge \neg(e_a \downarrow e_b)\}$	

Part I-Creation of sets. The rules R_1, R_2 and R_3 establish the creation of sets (Table I, Lines 24). An event which satisfies one of these rules creates a new set, and it is by default associated to such set as its left endpoint. R_1 creates a new set $W(e)$ when an event e does not have causal history. R_2 creates a new set $W(e)$ when it is detected that e is concurrent with respect to another event e_b. R_2 ensures that when the pattern $e_a \downarrow (e||e_b)$ occurs, the event e will be associated to a different set W than the sets for e_a and e_b. Finally, R_3 creates a new set $W(e)$ when it is detected that two concurrent events $e_a||e_b$ converge to a same event e. R_3 ensures that when the pattern $(e_a||e_b) \downarrow e$ occurs, the event e will be associated to a different set W than the sets for e_a and e_b.

Part II-Association of events. An event does not accomplish any of the rules of Part I, then it will be associated to an existing set W according to the rule R_4 (Table I, Line 6). R_4 associates events to sets by respecting the specification of the ordered set of events previously presented where the elements of a set compose a linearization based on the IDR. Each new event associated to a set W becomes its right endpoint.

Part III-Arrangement of sets. The resulting sets $W \in S$ are arranged according to the IDR. We say that a pair of sets $X, Y \in S$ are IDR-related "$X \downarrow Y$" if the following condition is satisfied: $X \downarrow Y$ if $x^+ \downarrow y^-$.

III. PROPOSED APPROACH

The general structure of our approach, presented in Figure 1, shows three main transitions: the VCs2HBR for the generation of the causal graph (HBR graph), the HBR2IDR for the generation of the IDR graph and the IDR2CAOS for the generation of the CAOS graph. Means that, we start with a database of Vector Clocks as input, from which we generate the HBR graph. Then, we generate the IDR graph from the HBR graph and the CAOS graph from the IDR graph. To do this, we are mainly based on the HBR, IDR and CAOS concepts. To implement our approach, we have used c++. The graph files are written in XML using the GraphML format. It consists of a language core to describe the structural properties of a graph. We are specially based on the tinyxml parser to read and create new xml files.

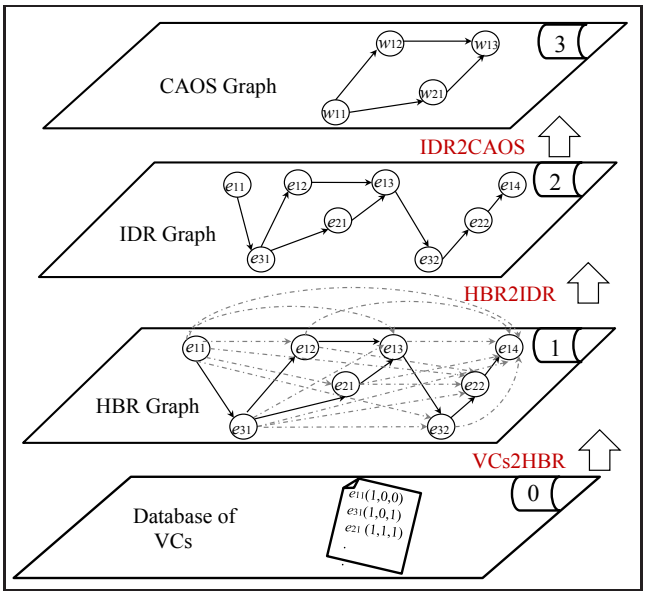

Fig. 1. The general architecture of our approach

A. VCs2HBR

Algorithm 1: VC2HBR

Input : E: The set of events
$VC(E)$: The set of vector clocks of E
Output: HBR: The HBR graph

```
1  buildHBR(VC)
2  begin
3      HBR ← ∅;
4      E_hbr ← ∅;
5      for each VC_id1 ∈ VC do
6          for each VC_id2 ∈ VC do
7              if (VC_id1 ! = VC_id2) then
8                  switch compare (VC_id1, VC_id2) do
9                      case "<"
10                         E_hbr ← E_hbr ∪ {(id1, id2)}
11                     case ">"
12                         break;
13                     case "not comparable"
14                         break;
15     for each element (id1, id2) ∈ E_hbr do
16         if (∃id ∈ E and ∃ (id1, id) ∈ E_hbr and
           ∃(id, id2) ∈ E_hbr) then
17             (id1, id2).label ← "t";
18         else if (id1.proc = id2.proc) then
19             (id1, id2).label ← "c";
20         else
21             (id1, id2).label ← "d";
22     Return HBR = (E, E_hbr)
```

VCs2HBR algorithm has as input a database of Vectors Clocks (VCs) generated over a distributed system by some causal algorithm, where each VC corresponds to a relevant event executed in such a system. An example of a database of VCs is given in Figure 2. In this example, the system is composed of three processes and eight events. The HBR edges are constructed by comparing each vector VC to the rest of the vectors (Algorithm 1, Lines 4-13). Fonction *compare* allows the comparison of each pair of vecteur clocks in a given set of VCs according to Definition 4. Means that, if the vector VC of an event a ($VC(a)$) is less than the vector VC of an event b ($VC(b)$), then an edge is generated between the corresponding nodes from a to b (Algorithm 1, Lines 8-9). In Figure 2, we

show an example of comparing the vector $VC(e_{11})$= (1, 0, 0) to the rest of the vectors. Since $VC(e_{11})$ is less than the others, an HBR edge is generated from e_{11} to each node of the rest of the events. The graph to the right is the HBR graph of the system. It contains three different labeled edges corresponding to the three condition of the HBR (Algorithm 1, Lines 15-21):

- c-labeled edge: a local relation which connects two successive events belonging to the same process.
- d-labeled edge: a direct relation between two events belonging to different processes (*i.e.* in case of communication).
- t-labeled edge: a transitive relation.

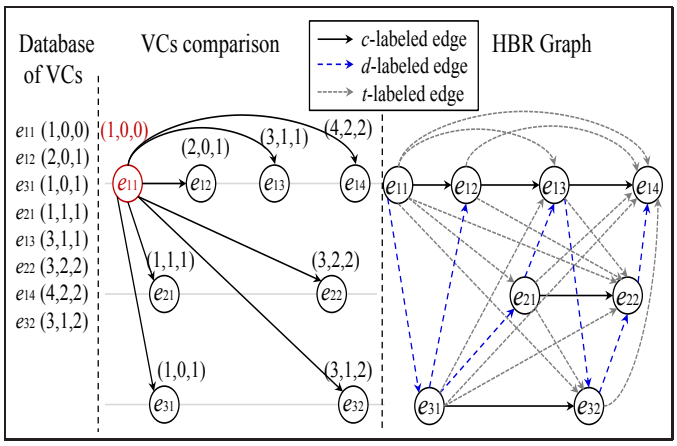

Fig. 2. An example of the generation of the HBR graph

B. HBR2IDR

Algorithm 2: HBR2IDR

Input : HBR: The HBR graph
Output: IDR: The IDR graph

```
1  buildIDR(HBR)
2  begin
3      IDR ← ∅;
4      E_idr ← E_hbr ;
5      for each (id_1, id_2) ∈ E_hbr do
6          verify(id_1, id_2) ;
7      Return IDR = (E, E_idr)
8  verify(id_1, id_2)
9  begin
10     if ∃ (id_1, id) ∈ HBR) and ∃((id, id_2) ∈ HBR) then
11         E_idr ← E_idr \ {(id_1, id_2)} ;
```

HBR2IDR algorithm has as input the set of HBR nodes and egdes. To obtain the IDR graph, HBR2IDR eliminates all the edges in the HBR graph which do not satisfy the IDR condition (an event a immediately-precedes an event b if it does not exist an event $c \in E$, such that c belongs to the causal future of a and to the causal past of b). The Fonction $verify$ is to verify which event in the set of HBR edges " E_{hbr}" satisfy such condition (Algorithm 2, Lines 6-9). Therefore, only immediate edges are kept. An example of executing HBR2IDR algorithm to the HBR graph of Figure 2 is given in Figure 3. The graph to the right is the IDR graph of the system.

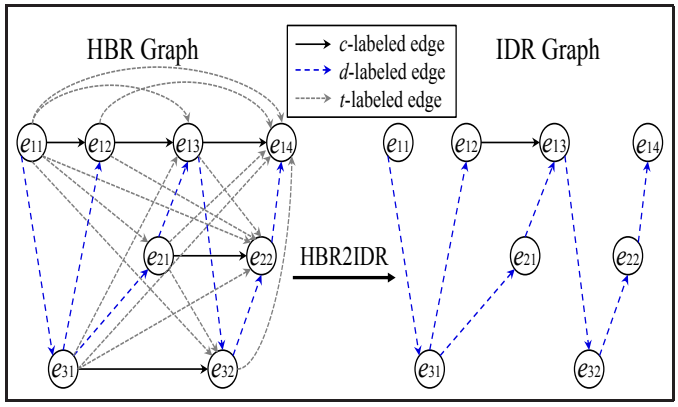

Fig. 3. An example of the generation of the IDR graph

C. IDR2CAOS

IDR2CAOS algorithm has as input the set of IDR nodes and egdes. It is based on the CAOS rules to generate the CAOS graph from the IDR graph. Firstly, it start with creating sets (Algorithm 3, Lines 7-15). Therefore, it verifies which event in the IDR nodes can create a new set (ie which event satisfies one of the rules R_1, R_2 and R_3 (Table I, Lines 1-3). R_1 creates a new set W_{id} when an event id does not have causal history. Means that, there is no edge (id_2, id) belonging to the set of IDR edges "E_{idr}" (Algorithm 3, Lines 8-9). R_2 creates a new set W_{id} when the pattern $id_1 \downarrow (id||id_2)$ occurs (Algorithm 3, Lines 10-11). Finally, R_3 creates a new set W_{id} when the pattern $(id_1||id_2) \downarrow id$ occurs (Algorithm 3, Lines 12-13). Events which do not saisfy one of these rules forms the set Evt. Then, IDR2CAOS associates these events to the created sets according to rule R_4 (Table I, Line 6). Means that, it verifies if the endpoint of each set immediately precedes one of these events (Algorithm 3, Lines 16-18). Then, IDR2CAOS arranges sets (Algorithm 3, Lines 19-25). Thus, it compares the endpoints of each pair of sets and verifies if it exists an immediate relation connecting them.
Likewise, an example of executing IDR2CAOS algorithm to the IDR graph of Figure 3 is given in Figure 4. In this example, the event e_{11} satifies the rule R_1, the events e_{12}, e_{21} satify the rule R_2 and the event e_{13} satifies the rule R_3. The graph to the right is the CAOS graph of the system.

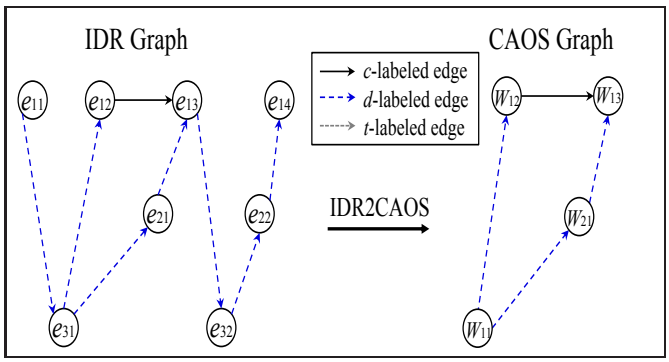

Fig. 4. An example of the generation of the CAOS graph

Algorithm 3: IDR2CAOS

Input : IDR: The IDR graph
Output: CAOS: The CAOS graph

```
1  buildCAOS(IDR)
2  begin
3      CAOS ← ∅;
4      E_caos ← ∅;
5      W → ∅;
6      Evt → ∅ ;
7      for each id ∈ ID do
8          if ∄(id_2, id)∈ E_idr) then
9              W ← (W_id);
10         else if ∃(id_1, id)∈ E_idr) and ∃(id_1, id_2)∈ IDR)
           then
11             W ← (W_id);
12         else if ∃(id_1, id)∈ E_idr) and ∃(id_2, id)∈ IDR)
           then
13             W ← (W_id);
14         else
15             Evt ← id ;
16     for each id ∈ Evt do
17         if ∃(id_1, id)∈ IDR) and W_id2[n] = id_1 then
18             W_id2 ← W_id2 ∪ {id} ;
19     for each W_id in W do
20         for each W_id2 in W do
21             if W_id2 != W_id then
22                 w+ ← W_id[0] ;
23                 w− ← W_id2[n] ;
24                 if ∃(w+, w−)∈ E_idr) then
25                     E_caos ← E_caos∪ {(W_id1, W_id2)} ;
26     for each element (id_1, id_2) ∈ E_CAOS do
27         if (id_1.proc = id_2.proc) then
28             (id_1, id_2).label ← "c";
29         else
30         (id_1, id_2).label ← "d";
31     Return CAOS = (W, E_caos)
```

IV. Case Study

In this section, we present a validation approach for checkpointing algorithms in which the use of causal graphs can be efficient. Checkpointing is a widely used solution to provide fault tolerance for distributed systems. Processes achieve fault tolerance by saving recovery information periodically during execution. When a failure occurs, the previously saved recovery information can be used to restart the computation from an intermediate state. The recorded states of a process called local checkpoint and the set of local checkpoints forms a global checkpoint or snapshot. We consider only the checkpoints as internal events. We denote by C_i^x the x^{th} checkpoint of process p_i. Finding when local checkpoints can be combined to form a consistent snaphot is a critical problem. 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 [3]. According to Netzer [6], a set of checkpoints, C, from different processes can belong to the same consistent global snapshot iff no checkpoint in C has a Z-path to any other checkpoint (including itself). Netzer [5] defines the notion of Z-path as follows:

Definition 6. *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.*

Property 3. *The length of a Z-path is l if the Z-path is formed by l messages.*

Definition 7. *A noncausal Z-path from a checkpoint to a checkpoint is a sequence of messages m_1, m_2, $\cdots$, m_n satisfying the conditions of Definition IV such that for at least one i $(1 < i < n)$, m_i is received by some process Pr after sending the message m_{i+1} in the same checkpoint interval.*

Definition 8. *A Z-cycle is a Z-path from a checkpoint to itself.*

To validate the correctness of checkpointing algorithms, we can model its execution by using causal graphs and then verify if in such graphs, the algorithm is exempt from dangerous patterns like Z-paths and Z-cycles. In this section, we are intersted on noncausal Z-paths detection. The general pattern of a noncausal Z-path in causal graphs is shown in Figure 5.

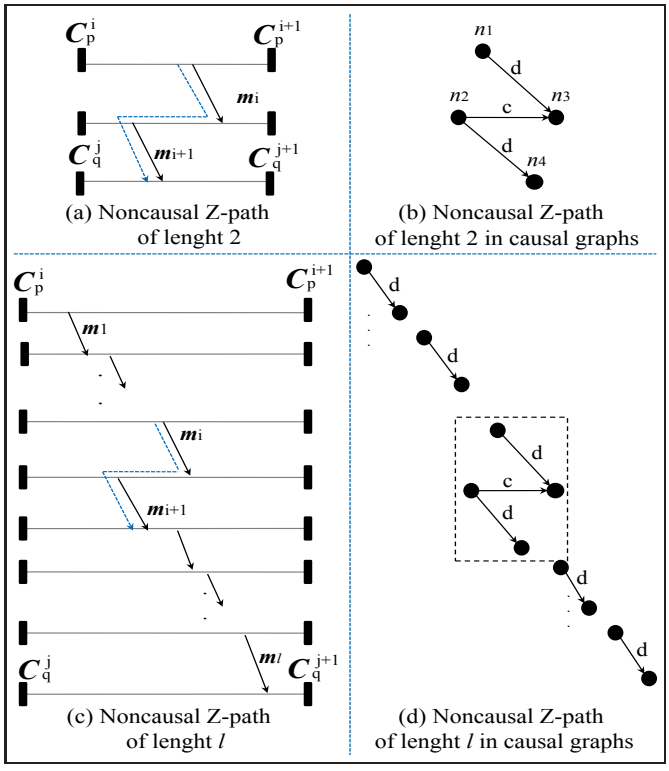

Fig. 5. General pattern of noncausal Z-path

A noncausal Z-path can be of length two or more. The Z-path of length two is in the form of ($n_1 \xrightarrow{d} n_3$, $n_2 \xrightarrow{c} n_3$, $n_3 \xrightarrow{d} n_4$). Why a Z-path is also of the form of d-c-d in a CAOS graph? In general, when we generate the CAOS graph from the IDR graph, the nodes (n_1, n_2, n_3, n_4) constructing a Z-path can not be regrouped in a same set. In fact, the node n_4 satisfies the second rule of the CAOS rules (Table I, Line 2) means that the node n_4 will be associated to a different set than the sets for n_2 and n_3. Added to that, the node n_3 satisfies the second rule of the CAOS rules (Table I, Line 3) means that the node n_3 will be associated to a different set than the sets for n_1 and n_2 (see the general pattern of a Z-path of length two in Figure5-(b)). Consequently, the form d-c-d is also conserved in the generated CAOS graph. A Z-path of length more than two is a succession of messages (d-labeled edges) and there exists at least two successive messages m and $m+1$ such that delivery(m) $\rightarrow$ send($m+1$), that means the general pattern of a noncausal long Z-path contains at least one occurrence of the pattern d-c-d. To detect noncausal Z-paths in a causal graph, we just need to verify if it contains this pattern. Algorithm 4 has as input a causal graph. It allows the detection of noncausal z-paths in such a graph. The idea is to add a z-labeled edge once the pattern d-c-d is detected (Algorithm 4, Line 11-12). Consequently, based on the graph output, we can decide if the algorithm is exempt of noncausal Z-paths.

Algorithm 4: Z-path-Detect

```
   Input  : G_in = (Node, Edge): The causal graph
   Output: G_out: The graph output
1  detect (G_in)
2  begin
3      G_out ← ∅ ;
4      for each (id_1, id_2)∈ Edge do
5          if ((id_1, id_2).label="d") then
6              test((id_1, id_2)) ;
7      Return G_out = (Node, Edge)
8  test(id_1, id_2)
9  begin
10     if (∃(id_3, id_2) ∈ Edge) and ∃(id_3, id_4) ∈ Edge) and
       (id_3, id_2).label="c" and (id_3, id_4).label="d") then
11         (id_1, id_4).label = "z" ;
12         Edge ← Edge ∪ {(id_1, id_4)} ;
```

In Figure 6, we give an example to show the detection of non causal z-path in the HBR, IDR and CAOS graphs. In this example, it exists a Z-path of length three from C_1^2 to C_4^2. It is denoted by the addition of a z-labeled edge.

V. Conclusion and Future Work

In this paper, we have presented a mechanism which constructs, for a distributed computation, the causal graph (HBR graph), the minimal causal graph (IDR graph) at a single event level and a causal consistent compact graph (CAOS graph) at an ordered event set level. The CAOS graph construction is very efficient for large-scale distributed systems as it drastically reduces the state-space of a system execution. Such a representation can be a solution to deal with other systems

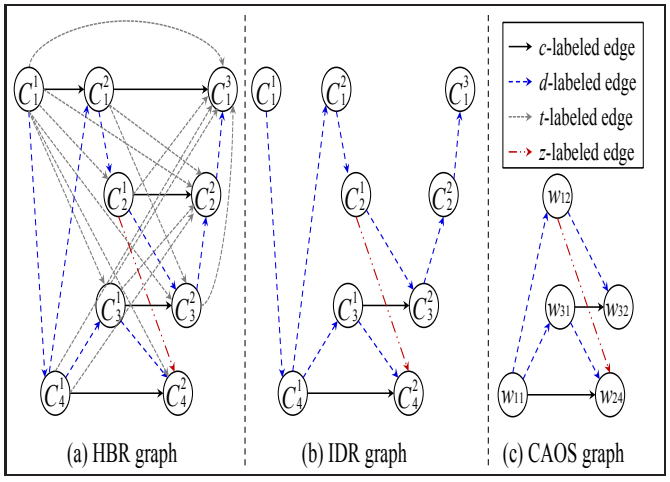

Fig. 6. Example of detecting Z-paths in the causal graphs

requirements such as checkpointing, debugging and so on. A case study is presented in this paper to illustrate how the causal graphs can be used for the validation of checkpointing algorithms.

As an on-going work, we aim applying our mechanism to large-scale systems to show the gain in terms of nodes and edges using the CAOS representation. as a future work, we propose providing a formal proof for the previous algorithms to prove their correctness and their efficiency.

References

[1] E.M Clarke, O. Grumberg, M. Minea, and D. Peled. State space reduction using partial order techniques. *International Journal on Software Tools for Technology Transfer, (STTT)*, 2(3):279–287, 1999.

[2] Colin J. Fidge. Timestamps in message-passing systems that preserve the partial ordering. *Proceedings of the 11th Australian Computer Science Conference. K. Raymond (Ed.)*, 10(1):56–66, February 1988.

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

[4] F.F. Mattern. Virtual time and global states of distributed systems. In *Proceedings of the International Workshop on Parallel and Distributed Algorithms*, pages 215–226, October 1988.

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

[6] 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.

[7] S. E. Pomares-Hernandez, J. R. Perez Cruz, and M.Raynal. From the happened-before relation to the causal ordered set abstraction. *Journal of Parallel and Distributed Computing*, 72(6):791–795, February 2012.

[8] S. E. Pomares-Hernandez, J. Fanchon, and K. Drira. *The Immediate Dependency Relation: An Optimal Way to Ensure Causal Group Communication*, volume 6, chapter 3 of the Annual Review of Scalable Computing, pages 61–79. Singapore University Press and World Scientific Publications, 2004.

[9] S. E. Pomares-Hernandez, J. Estudillo Ramirez, L.A. Morales Rosales, and G. Rodriguez Gomez. An intermedia synchronisation mechanism for multimedia distributed systems. *International Journal of Internet Protocol Technology*, 4(3):207–2018, 2009.

[10] K. Shimamura, K. Tanaka, and M. Takizawa. Group communication protocol for multimedia applications. In *Proceedings of the IEEE International Conference on Computer Networks and Mobile Computing, ICCNMC*, pages 303–308, 2001.