MCSR: A graph transformation based approach for Minimal and Compact Set Representation of Causal Dependencies in Distributed Systems*

Houda Khlif[1,*], Hatem Hadj Kacem[1] and Saul Pomares Hernandez[2,3]

[1]ReDCAD Laboratory, ENIS, University of Sfax, Tunisia
[2]INAOE, Tonantzintla, Mexico
[3]CNRS, LAAS, F-31400 Toulouse, France

Abstract

Causal ordering is an important property in distributed systems. Several algorithms have been developed over this principle. For example, there are solutions for roll-back recovery, multimedia synchronization, model checking, and many others. All these algorithms establish causal dependencies according to the Happened-Before Relation (HBR), which was introduced by Lamport, and they identify and ensure causal dependencies based on the Vector Clocks' algorithm introduced by Mattern and Fidge. The HBR is a strict partial order, and therefore, one main problem linked to it is the combinatorial state explosion. In this paper we present a graph transformation based approach called MCSR which constructs, for a distributed computation, the minimal causal graph (IDR graph) at a single event level and a causal consistent compact graph (CAOS graph) at an event set level. For this, we use the Immediate Dependency Relation (IDR) and the Causal Order Set Abstraction (CAOS), respectively. Both resultant causal graphs drastically reduce the state-space of a system. The MCSR receives a database of vector clocks as input, from which it automatically generates the fully-causal HBR original graph, the IDR graph, and the CAOS graph by using graph transformation rules. The resultant causal graphs can be used for different purposes, such as for the design of more efficient algorithms, validation, verification of the existing ones, among others. Finally, we compare the IDR graph and the CAOS graph with the HBR graph in terms of their number of nodes and edges.

Keywords

Distributed system, Causal Ordered Set Abstraction, Happened Before Relation, Immediate Dependency Relation, graph transformation.

1. Introduction

Causal ordering is an important property in distributed systems. The use of causal ordering provides built-in message synchronization, reduces the non-determinism 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. Several algorithms have been developed over the causal ordering principle. For example, there are solutions for checkpointing [1, 2],

TACC 2023: Tunisian-Algerian Joint Conference on Applied Computing, November 6 - 8, Sousse, Tunisia
*Corresponding author.
houda.khlif@redcad.org (H. Khlif); hatem.hadjkacem@redcad.org (H. Hadj Kacem); spomares@inaoep.mx (S. P. Hernandez)

roll-back recovery [3], predicate detection [4], group communication [5], multimedia data synchronization [6, 7], model checking [8], and many others. All these algorithms establish causal precedence dependencies according to the Happened-Before Relation (HBR), which was introduced by [9], and they identify and ensure causal dependencies based on the Vector Clocks (VCs) algorithm introduced by Mattern [10] and Fidge [11]. The HBR and VCs principles avoid the use of any kind of global references and synchronization mechanisms. The HBR is a strict partial order (transitive, irreflexive and antisymmetric) and therefore, one main problem linked to it is the combinatorial state explosion [12]. This characteristic of the HBR renders the analysis, verification and modeling of the existing solutions difficult, as well as the design of new more efficient algorithms. In this paper we present an approach called MCSR (Minimal and Compact Set Representation) which identifies, for a distributed computation, the minimal causal graph (IDR graph) at a single event level and a causal compact causal set graph (CAOS graph) at an event set level. In a previous work[13], we have used the algorithmic approach for the generation of HBR, IDR and CAOS. In this contribution, we adopt the graph transformation approach which offer an intuitive way to express and execute graph reductions, especially for complex relationships or patterns within the graph. In Section 4.2 a comparative is given between algorithmic solutions versus graph transformation. Our work use the Immediate Dependency Relation (IDR) [14] and the Causal Order Set Abstraction (CAOS) [15]. We start with a database of Vector Clocks as input, from which we generate the fully-causal HBR original graph. Then, by using graph transformation rules, we generate the IDR and the CAOS graphs. The efficiency of generating such graphs is shown by comparing them with the HBR graph in terms of their number of nodes and edges. As we will show, both resultant causal graphs drastically reduce the state-space of a system. The results show that the IDR and CAOS graphs use significantly fewer graphs components in order to represent the causality of the 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 show how causal graphs can be used for validation purposes.

The rest of the paper is structured as follows: Section 2 presents the system model and associated definitions. Our approach is detailed in Section 3, while some results are given in Section 4. Finally, Section 5 concludes with a few remarks.

2. Background and Definitions

2.1. 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 $e_{p,x}$.

2.2. Happened-Before Relation (HBR)

The Happened Before Relation for single events was defined by Lamport [9]. 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 it satisfies the following condition:

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

2.3. Vector Clocks

The Vector Clocks' algorithm was simultaneously developed by Fidge [11] and Mattern [10]. 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, the process p_j 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 between a pair of events in a system can be established as follows:

Definition 2. *For a pair of events $a, b \in E$, the event a causally precedes the event b if the following condition is satisfied:*

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

2.4. Immediate Dependency Relation (IDR)

The IDR is the transitive reduction of the HBR ([14]). It is denoted by "$\downarrow$", and it is defined as follows:

Definition 3. *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 belonging to E exists, such that c belongs to the causal future of a and to the causal past of b.

Property 1. *For all pair of events $a, b \in E$, $a \neq b$:*

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

2.5. Causal Ordered Set Abstraction (CAOS)

Assuming the poset E = (Ê,→) as the model adopted for a distributed computation, the objective of CAOS is to establish over this poset the rules and conditions of association of events and the conditions of ordering between the resulting sets ([15]).

Ordered sets of events. 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:

Part I-Creation of sets. The rules R_1, R_2 and R_3 establish the creation of sets (Table 1, Lines 2–4). 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. The rule R_1 creates a new set $W(e)$ when an event e does not have causal history. The rule R_2 creates a new set $W(e)$ when it is detected that e is concurrent with respect to another event e_b. The rule 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, the rule 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.

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
$W(e) \leftarrow R_4$. $W(e) \cup \{e' : \exists e_a \in W(e), \exists e_b \in E; e_a \downarrow e' \wedge \neg(e_a \downarrow e_b)\}$	6

Table 1
CAOS specification

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 1, Line 6). The rule 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. We say that a pair of sets $X, Y \in S$ are IDR-related "$X \downarrow Y$" if the following condition is satisfied:

$$X \downarrow Y \text{ if } x^+ \downarrow y^-$$

2.6. Graph Transformation

Graph Transformation is a rule-based approach targeted towards the modification on a graph ([16, 17]). A Graph 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 with R.

2.6.1. The Single PushOut Approach (SPO)

A rule of type SPO is a production in the form of (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 removing the graph corresponding to $Del = (L \backslash (L \cap R))$ and adding the graph corresponding to $Add = (R \backslash (L \cap R))$. According to the SPO approach, all the dangling edges are deleted.

2.6.2. The Double PushOut Approach (DPO)

A rule of type DPO is a production in the form of (L, K, R) where K is used to clearly specify the part to preserve after applying the rule. The difference between the SPO and DPO approaches is that the application of this rule is attached to the "Dangling Condition". This condition specifies that the rule can not be applied if its application will lead to suspended edges. If both conditions of existence of the occurrence of L and absence of suspended edges are satisfied, then 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 $Add = (R \backslash K)$.

2.6.3. Negative Application Condition (NACs)

In some cases, it is necessary to express additional conditions to specify conditions related to the absence of an occurrence in the graph. This type of condition is called Negative Application Conditions or NACs. An SPO rule has the structure (L, R, NAC) and is applicable to a graph G if there is an occurrence of L in G and if there is no case of NAC in G.

2.6.4. Neighborhood Controlled Embedding Approach (NCE)

The Neighborhood Controlled Embedding (NCE) mechanism ([17]) is based on the specification of the so-called connection instructions to allow the gluing of the added nodes to the neighbors of the removed nodes. These instructions are based on the labels of nodes to define new edges that will be introduced. Connection instructions are described by a pair (n, δ). The execution of this connection instruction involves the introduction of an edge between the added node n and all the neighbors nodes of the removed node whose label is δ.

dNCE (d for edge label), eNCE (e for edge label), and edNCE are extensions of the NCE approach.

The edNCE approach is the combination of eNCE and dNCE approaches. The edNCE connection instructions are in the form of $(n; p/q; \delta; d; d')$. Its execution implies the introduction of an edge in the direction indicated by d' between the node n and all nodes n' which are p-neighbours and d-neighbours (in-neighbours if d=in and out-neighbours otherwise) of the removed node.

3. Graph transformation based approach

The genaration of the IDR and CAOS graphs can be done with using graph transformation approaches. To do this, we have designed two sets of rules: the HBR2IDR for the generation of the IDR graph and the IDR2CAOS for the generation of the CAOS graph (Figure 1).

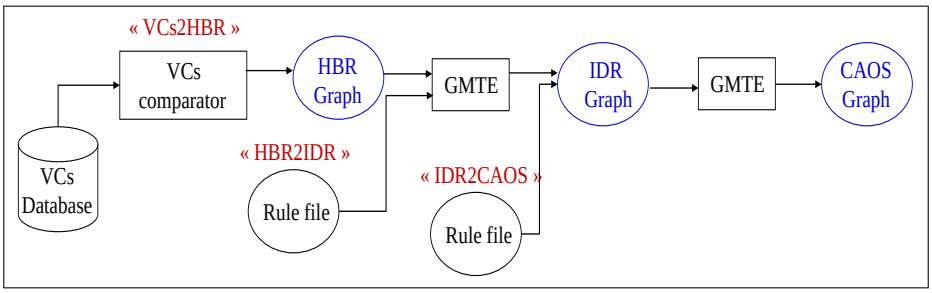


Figure 1: General architecture for GT based approach

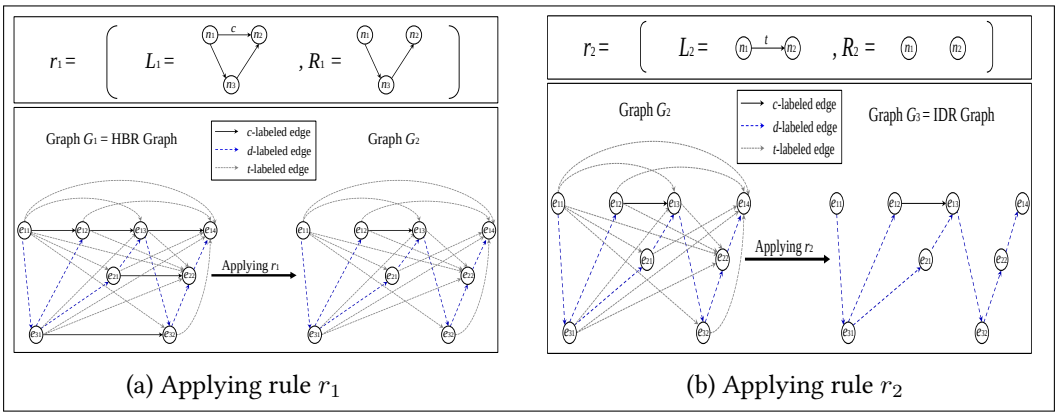
Our solution is based on two main approaches which are SPO and edNCE. The SPO approach offers a flexible way of dealing with dangling edges. The edNCE approach is for the addition and the removal of nodes. We have used GMTE[1](the Graph Matching and Transformation Engine) to implement our approach.

[1]GMTE and our MCSR approach are available at http://homepages.laas.fr/khalil/GMTE/

3.1. HBR to IDR

We have designed the HBR2IDR rules: r_1 and r_2 (presented in Figure 2a), which are sequentially executed. Rule r_1 verifies if it exists, in the HBR graph, local (c-labeled) edges which do not satify the IDR condition. Then, it deletes the existing ones. Rule r_2 deletes all transitive (t-labeled) edges from the HBR graph as they are not immediate.

Firstly, rule $r_1 = (L_1, R_1)$ is applied to the HBR graph. Its application implies finding all matches of L_1 in G_1 and replacing them with R_1. As a result, all c-labeled edges which are not immediate are removed. In fact, the c-labeled edge ($n_1 \xrightarrow{c} n_3$) in the pattern L_1 is not immediate because it exists a node n_3 such that $n_1 \rightarrow n_3 \rightarrow n_2$. An illustrative example is given in Figure 2a. In this example, it exists four matches of L_1. Therefore, the local edges $e_{11} \xrightarrow{c} e_{12}$, $e_{13} \xrightarrow{c} e_{14}$, $e_{21} \xrightarrow{c} e_{22}$ and $e_{31} \xrightarrow{c} e_{32}$ are removed and we obtain the graph G_2.



(a) Applying rule r_1 (b) Applying rule r_2

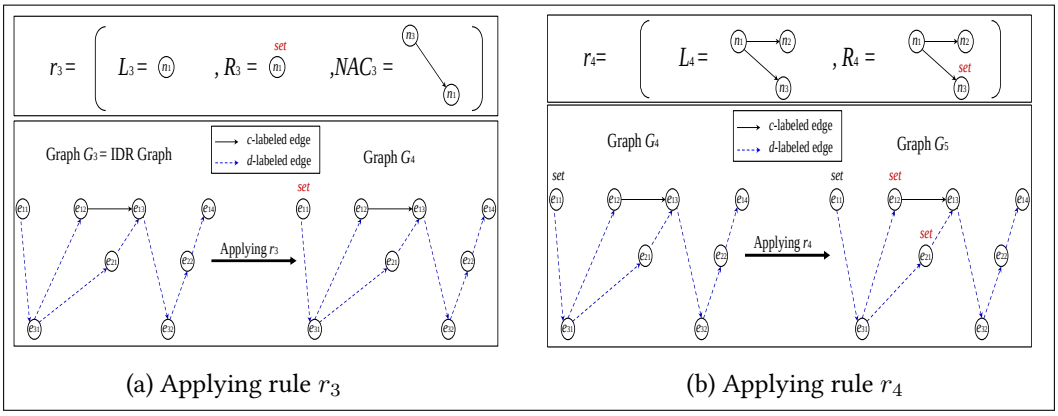
Figure 2: HBR to IDR rules

Then, rule $r_2 = (L_2, R_2)$, presented in the top of Figure 2b, is applied to G_2. Its application implies finding all maches of L_2 in G_2 and replacing them with R_2. That means finding all t-labeled edges and removing them. At this level, it is not necessary to verify if a transitive edge is immediate. It is clear in its definition (If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$) that the transitive edge does not satisfy the IDR condition. The result of r_2 application is shown in Figure 2b. The obtained graph G_3 is the IDR graph of the system.

3.2. IDR to CAOS

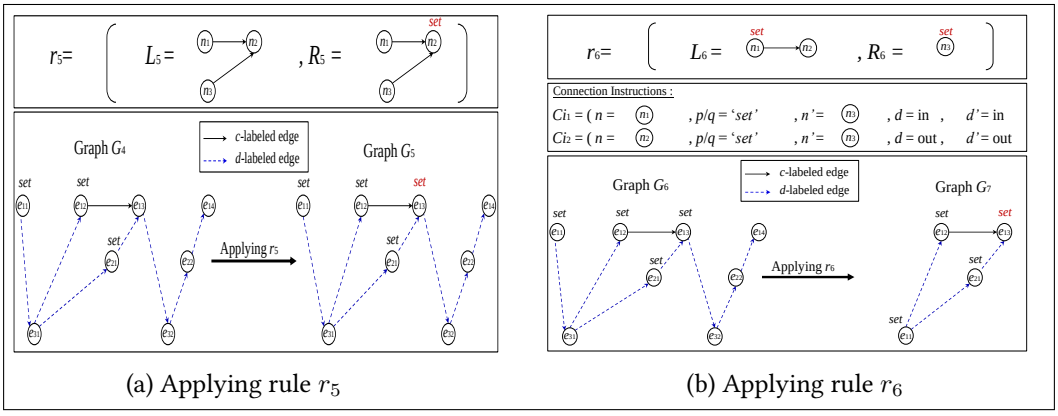
In order to arrange the events of the IDR graph into ordered sets, based on CAOS abstraction, we have designed the IDR2CAOS rules: r_3, r_4, r_5 for sets creation and r_6 for events association. These rules are also sequentially executed.

Initially, $r_3 = (L_3, R_3, NAC_3)$, presented in Figure 3a, is applied to the IDR graph. It verifies the first rule of the CAOS rules (Table 1, Line 2). It means that, a node n_1 creates a new set if it does not exist a node n_2 belonging to the causal past of n_1. This condition is satisfied with using the negative application condition $NAC_3 : L_3$ is replaced with R_3 if NAC_3 does not



(a) Applying rule r_3 (b) Applying rule r_4

Figure 3: IDR to CAOS rules (Part I)



(a) Applying rule r_5 (b) Applying rule r_6

Figure 4: IDR to CAOS rules (Part II)

exist. A node that satisfies such a condition is denoted by the label "set". In the given example, only the node e_{11} satisfies NAC_3 (there is no node which precedes e_{11}). e_{11} creates a new set. It is denoted by a label "set". The rule $r_4 = (L_4, R_4)$, presented in Figure 3b is nextly applied to G_4. It models the second rule of the CAOS rules (Table 1, Line 3). Its application implies, if the pattern L_4 exists, it will replaced by R_4. That means, the node n_2 creates a new set denoted by the label "set". In the used example (Figure 3b), two matches exit ($e_{31} \rightarrow e_{12}$, $e_{31} \rightarrow e_{21}$) and ($e_{31} \rightarrow e_{21}$, $e_{31} \rightarrow e_{12}$). Consequently, in the output graph G_5, the nodes e_{12} and e_{21} are denoted by a label "set".

The rule $r_5 = (L_5, R_5)$ is applied to graph G_5 (see Figure 4a). This rule corresponds to the third rule of the CAOS rules (Table 1, Line 4). Its application implies, if the pattern L_5 exists in G_5, it will replaced by R_5. Consenquently, the node n_2 creates a new set that we denote also by a label "set". Figure 4a shows the result of r_5 application. In this example, only one match exists ($e_{12} \rightarrow e_{13}$, $e_{21} \rightarrow e_{13}$) means that e_{13} creates a new set, therefore, it is denoted by a label "set". Finally, $r_6 = (L_6, R_6)$ is to associate the rest of events to those which have created a new set (set-labeled nodes). This rule is based on the edNCE approach (see section 2.6.4). Rule r_6 is

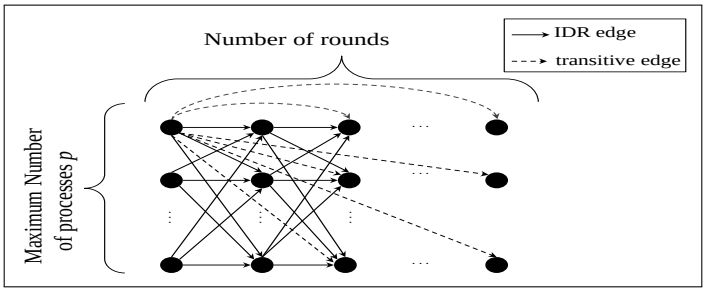


Figure 5: Maximum number of edges in an HBR graph

also of type SPO. its application implies deleting each occurence of L_6 in G_6 and replacing it with R_6. Consequently, r_6 removes two consecutive nodes $n_1, n_2 \in G_6$, adds a new node n_3 (set-labeled node) and finally, adds new edges to connect the added node to the neighbours of deleted nodes. The addition of edges is made according to the connection instructions Ci_1 and Ci_2, shown in the middle of Figure 4b. Ci_1 adds a new edge connection from the added node n_3 to a neighbour of the deleted node n_1. Ci_2 adds a new edge connection from each neighbour of the deleted node n_2 to the added node n_3. Rule r_6 is presented and illustrated with an example in Figure 4b. A first match regroupes e_{11} and e_{31} in one set. Three others matches exist e_{13} and e_{32}, the added node regrouping e_{13} and e_{32} (set-labeled node) and e_{22}, the new added node and e_{14}, respectively. G_7 is the CAOS graph of the system.

4. Evaluation Results

The objective of this section is to illustrate the viability of the IDR graph and CAOS graph construction and their efficiency in the causal dependency representation.

4.1. Why the IDR and the CAOS graphs are efficient?

Let p and n be two integers (p = number of processes in the system, n = number of events). The maximum number of edges in the HBR graph and the IDR graph is when we have the maximum number of concurrent messages in each round (see Figure 5). Therefore, the number of nodes is n = number of rounds $\times$ p.
The maximum number of edges (HBR edges = IDR edges + transitive edges) is:

$$e = p(n\text{-}p) + \frac{(n-p)(n-2p)}{2}$$

The maximum number of edges in the IDR graph is only:

$$e' = p(n\text{-}p)$$

The maximum number of edges in the CAOS graph is equal to the maximum number of edges in the IDR graph. In fact, in the worst case the number of edges in the CAOS graph generated from the IDR graph remains the same. Therefore, there is an important reduction of edges from the HBR graph to the IDR/CAOS graph. Actually, there is also a reduction of nodes and edges

from the IDR graph to the CAOS graph. To show this, we have generated by simulation, the Vector Clocks of three different system executions with 8, 21 and 50 events (nodes) and with 3, 8 and 15 processes, respectively. The events were triggered assuming a uniform distribution. For each execution, we have constructed the HBR graph, the IDR graph and the CAOS graph, and we compared the resulting graphs in terms of their number of nodes and edges. The obtained results are presented in Figure 6.

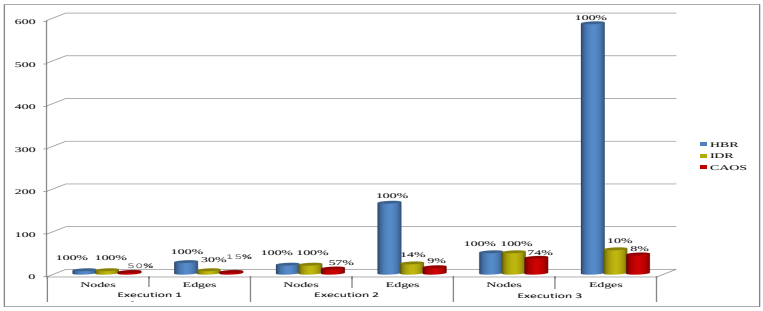


Figure 6: HBR, IDR and CAOS comparison

The results show that the number of edges is greatly reduced from the HBR graph to the IDR graph, and that the number of edges and nodes is even more reduced from the HBR graph to the CAOS graph. Likewise, there is an important reduction of edges and nodes from the IDR graph to the CAOS graph. The average reduction in the number of edges from the HBR graph to IDR graph and to the CAOS graph is about 82% and 89%, respectively. The average reduction in the number of nodes from the HBR graph and the IDR graph to the CAOS graph is about 41%, Finally, the average reduction in the number of edges from the IDR graph to the CAOS graph is about 36%. We conclude this analysis with the following remark. It is interesting to see that the reduction performance in the average number of edges in the IDR graph and the CAOS graph increases as the number of nodes in the HBR graph increases.

4.2. Advantages of using the graph transformation approach

Graph transformation approaches are widely used in various research fields to model and analyze complex systems that can be represented as graphs [18, 19]. They are used for tasks like modeling software systems, studying social networks, optimizing control systems, planning transportation networks, managing business processes and designing robotic behaviors. These approaches offer a versatile way to model, analyze, and understand complex systems in various disciplines. Graph transformation techniques can significantly reduce the time required to modify or analyze graphs. they provide a powerful and intuitive framework for modeling, analyzing, and managing the complexities of large-scale systems across various research domains. Their flexibility, formal analysis capabilities, and compatibility with modern software engineering principles make them well-suited for addressing the challenges posed by such systems. By defining transformation rules, complex modifications can be applied systematically and automatically to large graphs. This eliminates the need for time-consuming modifications and allows for efficient updates or transformations. Graph transformation techniques can handle large-scale graphs efficiently. By

leveraging rule-based transformations, the complexity of graph modifications or analyses can be managed effectively, allowing for scalability. This scalability ensures that the time cost remains manageable even for complex graph structures and large datasets. representing distributed systems as large graphs offers a powerful means to understand, analyze, and model the system's architecture, behavior, and interactions. In our case, we need to generate IDR and CAOS graphs from the HBR graph which is very large graph. Graph transformation approaches have several advantages when it comes to reducing a graph compared to using algorithmic solutions:

- They offer an intuitive way to express and execute graph reductions, especially for complex relationships or patterns within the graph.
- Graph transformation excels in pattern matching, making it efficient for identifying and reducing specific structures in the graph.
- Rule-based reductions simplify the reduction process and ensure transparency and maintainability.
- Graph transformation can leverage parallelism for large graphs.

When appropriately applied, graph transformation can provide powerful insights and solutions in a wide range of domains. These advantages stem from the inherent ability of graph transformation to represent complex structures in a flexible and intuitive way.

5. Conclusion

In this paper, we have presented a graph transformation based approach that 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. we have also illustrated the viability of the IDR graph and the CAOS graph construction and their efficiency in the causal representation. The results show that both resultant causal graphs drastically reduce the state-space of a system execution. On average the number of edges from the HBR graph to IDR graph and the CAOS graph was reduced by 82% and 89%, respectively, and the number of nodes from the HBR graph to CAOS graph was reduced by 41%. It is interesting to see that the performance reduction in the average number of edges in the IDR graph and CAOS graph increases as the number of nodes in the HBR graph increases.

References

[1] J. Helary, R. Netzer, M. Raynal, Consistency issues in distributed checkpoints, IEEE Transactions on Software Engineering 25 (1999) 274–281.

[2] A. C. Simon, S. E. Pomares-Hernandez, J. R. P. Cruz, A delayed checkpoint approach for communication-induced checkpointing in autonomic computing, in: 22th IEEE International WETICE Conference, 2013, pp. 56–61.

[3] R. Baldoni, J. Helary, A. Mostefaoui, M. Raynal, A communication-induced checkpointing protocol that ensures rollback-dependency trackability, in: Symposium on Fault-Tolerant Computing, 1997, pp. 68–77.

[4] P. Chandra, A. Kshemkalyani, Data-stream-based global event monitoring using pairwise interactions, Journal of Parallel and Distributed Computing 68 (2008) 729–751.
[5] A. Nakamura, T. Tachikawa, M. Takizawa, Causally ordering group communication protocol, in: Proceedings of the International Conference on Parallel and Distributed Systems, 1994, pp. 48–55.
[6] K. Shimamura, K. Tanaka, M. Takizawa, Group communication protocol for multimedia applications, in: Proceedings of the IEEE International Conference on Computer Networks and Mobile Computing, ICCNMC, 2001, pp. 303–308.
[7] S. E. Pomares-Hernandez, J. E. Ramirez, L. M. Rosales, G. R. Gomez, An intermedia synchronisation mechanism for multimedia distributed systems, International Journal of Internet Protocol Technology 4 (2009) 207–2018.
[8] M. Wehrle, M. Helmert, The causal graph revisited for directed model checking, in: International Static Analysis Symposium, volume 5673 of *LNCS*, Springer, 2009, pp. 86–101.
[9] L. Lamport, Time, clocks and the ordering of events in a distributed system, Communications of the ACM 21 (1978) 558–565.
[10] F. Mattern, Virtual time and global states of distributed systems, in: Proceedings of the International Workshop on Parallel and Distributed Algorithms, 1988, pp. 215–226.
[11] C. J. Fidge, Timestamps in message-passing systems that preserve the partial ordering, Proceedings of the 11th Australian Computer Science Conference. K. Raymond (Ed.) 10 (1988) 56–66.
[12] E. Clarke, O. Grumberg, M. Minea, D. Peled, State space reduction using partial order techniques, International Journal on Software Tools for Technology Transfer 2 (1999) 279–287.
[13] H. Khlif, H. Hadj-Kacem, S. P. Hernandez, A. Hadj-Kacem, A mechanism for the causal ordered set representation in large-scale distributed systems, in: Proceedings of the 24nd International WETICE Conference, IEEE Computer Society, 2015, pp. 86–101.
[14] S. E. Pomares-Hernandez, J. Fanchon, K. Drira, The Immediate Dependency Relation: An Optimal Way to Ensure Causal Group Communication, volume 6, Singapore University Press and World Scientific Publications, 2004, pp. 61–79.
[15] S. E. Pomares-Hernandez, J. R. P. Cruz, M.Raynal, From the happened-before relation to the causal ordered set abstraction, Journal of Parallel and Distributed Computing 72 (2012) 791–795.
[16] J. Engelfriet, G. Rozenberg, Handbook of Graph Grammars and Computing by Graph Transformation, World Scientific Publishing, 1997, pp. 1–94.
[17] G. Rozenberg, Handbook of graph grammars and computing by graph transformation, World Scientific Publishing (1997).
[18] G. Fayçal, A. Chaoui, A graph transformation based approach for multi-agent systems reorganization, Multiagent Grid Syst. 15 (2019) 375–394.
[19] H.-J. Kreowski, S. Kuske, A. Lye, A. Windhorst, A graph-transformational approach for proving the correctness of reductions between NP-problems, Electronic Proceedings in Theoretical Computer Science 374 (2022) 76–93.