ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, Vol 8(33), DOI: 10.17485/ijst/2015/v8i33/75502, December 2015 ISSN (Online) : 0974-5645

Modeling Causal Consistency in a Distributed Shared Memory using Hierarchical Colored Petri Net

Shamim Yousefi*, Samad Najjar Ghabel and Leyli Mohammad Khanli

Faculty of Electrical and Computer Engineering, University of Tabriz, 29th Bahman Boulevard, Tabriz, Iran;
shamimyousefi75@gmail.com, samad.najjar@gmail.com, l-khanli@tabrizu.ac.ir

Abstract

Causal consistency is one of the consistency models that categorizes events in a distributed system to those causally related, and those are not. In this paper, a new modeling of the causal consistency checking in a distributed shared memory is presented using the hierarchical colored Petri net, and its rules and properties are modeled with all details. Our proposed model takes a scenario of the performed processes in the distributed shared memory system as an input and determines whether this scenario is allowed with a causally consistent shared memory system or not. Modeling of the causal consistency using the hierarchical colored Petri net enables to study the state space of a distributed shared memories system and automatically identify, which write operations are causally related and which of them are not. Furthermore, state space analysis of the proposed model shows that in the last node of the state space graph, all processes of the distributed shared memory system are finished or not. Therefore, it is represented that the model of the distributed shared memory system is reached to definitive state or not, and checks the safety of the shared memory system's current state. Because of these features, the proposed model is used in industrial activities, particularly the computer-based distributed systems. In fact, pre-production modeling and simulation of the industrial systems using the hierarchical colored Petri net will be economically and very affordable.

Keywords: Causal Consistency, Hierarchical Colored Petri Net, Modeling, Shared Memory Systems, Verification

1. Introduction

In distributed systems like distributed data stores or distributed shared memory systems like databases, consistency models are necessary for the harmony between shared memory and the processes[1–3]. Consistency models guarantee that if the processes follow specific rules on memory, the system will work property; and it supports the given model. In other words, Consistency models cause that if the programmers follow the determined rules, distributed shared memory will be consistent and the results of shared memory operations will be predictable[4–6]. Causal consistency is considered one of the weak models of the sequential consistency. It divides the processes into those causally related and those are not[7–9]. In this paper, a new model of the causal consistency checking in a distributed system is presented to use the hierarchical colored Petri net and its rules and properties are modeled with all details. Our proposed model takes a scenario of the performed processes in the distributed shared memory system as an input and determines whether this scenario is allowed with a causally consistent shared memory system or not. The modeling of the causal consistency enables us to study the state space of a distributed shared memory and automatically identify which write operations are causally related and which of them are not. In order to define the terms of petri net presented in the paper. According to the paper some related definitions are:

* Author for correspondence

Definition 1. Colored Petri Net is defined as a nine-tuple $CPN = (P, TD, A, \Sigma, V, C, G, E, I)$, where[10]:

- P: An exhaustible set of places.
- TD: An exhaustible set of transitions TD (such that $P \cap TD = \emptyset$).
- $A \subseteq (P \times T) \cup (TD \times P)$ is a set of arcs.
- Σ: An exhaustible set of color sets (non-Null).
- V: An exhaustible set of typed variables such that Type[v]Σ (for all variables v V).
- $C: P \rightarrow \Sigma$ indicates a color set function that each place is appointed by a color set.
- $G: TD \rightarrow EXPR_V$ indicates a guard function that each transition t such that Type[G (t)] = Bool is appointed by guard function.
- $E: A \rightarrow EXPR_V$ indicates an arc expression function that appointed an arc expression to each arc a such that $Type[E(a)] = C(p)_{MS}$, where p is the place connected to the arc a.
- $I: P \rightarrow EXPR_{\emptyset}$ indicates an initialization function that appoints an initialization expression to each place p such that $Type[I(p)] = C(p)_{MS.}$

Definition 2. A Colored Petri Net Module contain a four-tuple $CPN_M = (CPN, T_{sub}, P_{port}, PT)$, where[10]:

- CPN is a non-hierarchical Coloured Petri Net that was defined above in Definition 1.
- $T_{sub} \subseteq TD$ is a set of substitution transitions.
- $P_{port} \subseteq P$ is a set of port places.
- $PT: P_{port} \rightarrow \{IN, OUT, I/O\}$ is a port type function that appoints a port type to each port place.

Definition 3. A hierarchical Coloured Petri Net CPN_H, the following concepts are defined[10]:

- A compound place is a set of place instances related via port–socket relations or fusion sets.
- A hierarchical CPN models inclusive of a limited set S of modules. $\forall s \in S : s = \left(CPN^s, T^s_{sub}, T^s_{port}, PT^s\right)$.
- A marking M is a function that maps each compound place [p*] into a multi set of tokens , $M([p^*]) \in C(p)_{MS}$, where (p,s*) defines any place indicates dependent to [p*].
- The initial marking M_0 is appointed by M_0 ([p*]) = I(p) <> where (p,s*) defines any place instances dependent to [p*].

The rest of the paper is organized as following: A brief overview of related works is given. In Section 2, a case study of the causal consistency is discussed, and its features are explained. Section 3 includes the model of the causal consistency and the preliminary elements of the model. In Section 4, the functions of the causal consistency model are discussed, and their implementations are explained. In Section 5, the state space graph of the causal consistency model is analyzed. Finally, the paper is concluded in section 6.

Many different methods, mainly based on logical formalisms have been presented for modeling causal knowledge. Petri net is a powerful formal method for representational properties of the various distributed systems, and causes state space analysis. Therefore, Petri net is a useful tool for studying the properties of the causal consistency model for distributed systems. Portinale[11] presented an approach to causal consistency model in distributed systems in which its semantics was provided by Petri net. It is shown how the technique of modeling is strong enough to capture all important aspects of the causal model without resorting to complex structures. Furthermore, a complete simulation of the aspects concerning the accuracy of the represented causal model was provided in terms of accessibility in the Petri net. The analysis of the author verified the proposed model capable of discovering incorrectness by using analysis tools accessible for Petri net and the clear parallelism of the model.

Other various models for studying causal consistency in distributed systems using different categories of formal methods are developed in the paper. Brzezinski et al.[12] discussed the relationships between data-centric consistency models and client-centric consistency models. Furthermore, they used a consistent formula to present formal definitions of consistency models in the context of distributed shared objects. Authors proved that causal consistency should guarantee all common session. In other words, read-your-writes, monotonic-reads, monotonic-writes and writes-follow-reads are preserved in which Lloyd et al.[13] presented a model and implementation of COPS, a key-value store that provided causal consistency across the wide-area. The central procedure in COPS was tracking and perspicuity checking, whether causal dependencies between keys were satisfied in the local cluster before executing writes. Furthermore, in COPS-GT, it was introduced the transactions in order to get a consistent view of multiple keys without blocking or locking.

Some other works present a formal definition of causal memories and provide various implementations for message-passing systems. They describe a practical group of programs. If they are implemented in a strongly

consistent memory, they will run right with causal memory[14,15]. Furthermore, the authors presented some optimal protocols for causal consistency in distributed systems. Their proposed protocols were presented for partial replication in the distributed systems. Shen et al.[15] consider the complete replication as a special case of protocols and gives three optimized implementation of causal consistency for the replication case. These implementations are based on Full-Track, Opt-Track and Opt-Track-CRP algorithms.

Colored Petri net is a top-level Petri net which uses ML language[16]. This modelling tool has suitable capabilities that give more capacities to model and analyze different distributed systems. In this paper, we have gained a lot of advantages by modeling the causal consistency checking in distributed shared memory systems using hierarchical colored Petri net.

2. Case Study of model

Causal consistency model categorizes the processes to the causally related ones and those are not. In other words, in a distributed shared memory system considering causal consistency, the writings that are potentially causally related must be observed by all processes equally, while the writings that are concurrent may be observed in an unlike order on different machines. As a more accurate definition, if event B influenced or caused by an earlier event, A, causal consistency requires that every process first sees A, then B. However, if two processes simultaneously and spontaneously indicate different variables, they are not causally related together, and they are said to be concurrent. In the other words, assume that process P1 indicates the variable V1. Then P2 reads V1 and writes V2. Here the reading of V1 and the writing of V2 are potentially causally related, because when there is a reading followed by a writing, the two events are potentially related[1].

Table 1. This scenario is allowed by a causally consistent shared memory system[1]

P1	W(X) a			W(X) c		
P2		R(X) a	W(X) b			
P3		R(X) a			R(X) c	R(X) b
P4		R(X) a			R(X) b	
						R(X) c

In order to make the subject clearer, suppose that a distributed shared memory contains four different processes as shown in Table 1. This scenario is allowed by a causally consistent shared memory system. In proposed scenario, the reading of R(X) a and the writing of W(X) b by P2 are potentially causally related, and every process first sees R(X)a, then R(X)b. It is worth mentioning that W(X) b and W(X) c are concurrent. Therefore, each process is not required to see them in the same order.

3. Colour Sets, Initial Marketing and the Models of the System

Figure 1 shows the top-level model of the distributed system. In the proposed model, before entering the scenario as an input, all of shared variables and their initial values belong to *Shared variables* place. Furthermore, there are two other places in the top-level model with the names of *Global Causal Consist Order* and *Other not Causal Variables Value* that include the events that are causally related and those are not related, respectively. Definitions of colour sets that are used in the model of the system are like the following:

colset variable=STRING;
colset value=INT;
colsetP_id=INT;
colsetTs=INT;
*colsetcasuals_process=product P_id*value*Ts;*
colset casuals=list casuals_process;
*colset variables= product variable*value;*
*colsetL_var= product variable*P_id*value*Ts;*
*colsetg_var= product variable*casuals;*

The colour sets of *variable* and *value* are defined to represent the name of the variables that are used in the system model and their values. They are a set of all text strings and integers respectively. Each of the four processes in the designed distributes shared memory system is determined by a unique number. These numbers are allocated to the input packets. The colour set *P-id* is used for the unique numbers and they are a set of all integers. The colour set *Ts* is a set of all integers that is used to represent the time of the reading and writing. The Colour set *casuals_process* is defined to be represented when a process is read or written on the certain variable.

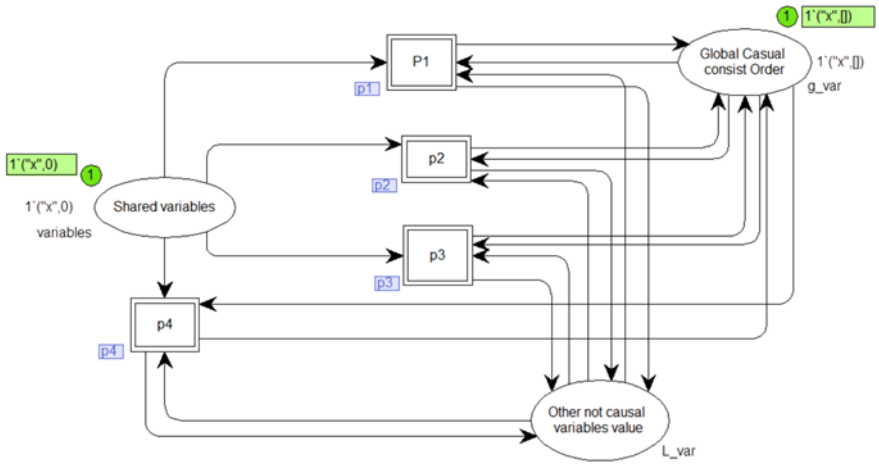

Figure 1. Top-level module of the hierarchical causal consistency model.

The colour set of *casuals* represent a list of the *casuals_process*. Every variable that is used in the proposed model has a name and a value that is represented by colour set *variables*. Colour set *L_var* determines which what operation on what variable is done. Colour set *g_var* represents a list that consists the actions they process performed on the variables.

Figure 2 shows the low-level sub-module of colored Petri net model for causal consistency checking in a distributed shared memory system for every process. It is worth mentioning that in the proposed model, there are four processes. The colour sets that are used in the sub-module models are:

colsetR_list=list variable;
colset UTC=INT;
*colsetcasuals_process_Backup=product variable*P_id*value*Ts;*
colsetRead_Backup_list=list casuals_process_Backu

Colour set *UTC* is defined to be equal to the set of all integers and it is used for representing the global clock of the distributed shared memory system. All processes should be synchronized by this global clock. Colour set *R_list* represents a list of the variables. Colour set *casuals_process_Backup* is defined to represent what values are read by each process, and a list of *casuals_process_Backup* is represented by colour set *Read_Backup_list*.

There is place in every sub-model through the name of *shared variables* for every process to use the Shared variables. In fact, this is a part of the place that includes the shared variables in the top-level module. There is a place called *Process ID* in every process sub-module that has the limited numbers of the processes. When a variable enters the sub-module, it takes the unique number of corresponding to the process along with time. Then the variable will be ready to be used as a local variable for the relevant module. Practically, in the following, the local variables of each process can be defined:

(Variable name, Process ID, Value, Time)

These variables are located in *Current Local Variable* places of each process. Finally, the functions and arcs' explanations check (that is described in the next section) the sequences for the causal consistency rules and puts them into the proper place. The final values of the variables are shown in the *Global Causal consist Order* and *Other not Causal Variables value* that are placed in the top-level module.

A case study approach was used to test the proposed model. Therefore, the initial marking and variables of the test CPN model are like the following:

---

var vars,vars1,v,v1,v2:variabvar write:INT;
var T,T1:Ts var R:R_list;
var B:Read_Backup_list; var L2:casuals_process_Backup;
var T,T1:Ts; var a,b,c,d,d1,a1,b1,a2,b2,d2:INT;
varRandom_BOOL:BOOL;var r:BOOL;
var L,L1:casuals;
valint_type=100;

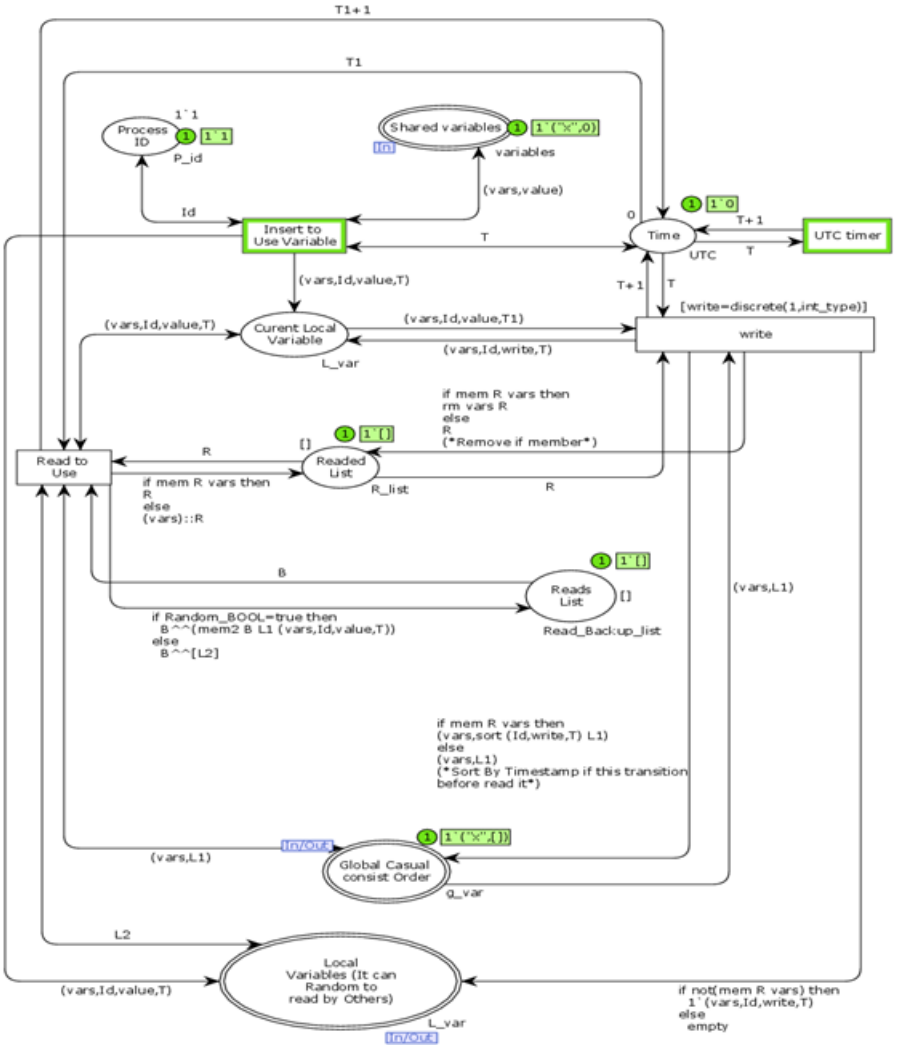

Figure 2. CPN model of process's sub-module instance corresponding to process P1.

4. Description of Model's Functions

Different functions using the ML language are used in the model for defining sort, checking causal consistency requirements, comparing and removing. Colored Petri net do not have directly powerful arcs, but they exist in some extended Petri nets[17]. In this section, all implemented functions in the hierarchical model are described.

Function *mem* takes a list of packets and a certain packet as parameters. This function checks the existence of a packet in the input list. If the list does not contain the desired packet, the function *mem*will return false; otherwise it will return true.

```
funmem [] a = false
        | mem (x::xs) a = a= x orelsememxs a;
```

Function *sort* takes a list of input packets as parameter and sorts them based on their input times. If the input times of several packets are the same, the function *sort* arrange them based on their process number.

```
fun sort(a,b,d) [] = [(a,b,d)] |
sort (a,b,d) ((a1,b1,d1)::queue) =
ifcheack (d,d1) then
        (a,b,d)::(a1,b1,d1)::queue
else
        (a1,b1,d1)::(sort (a,b,d) queue);
```

Function *mem2* is the main part of the protocol that checks the causal relation between the events. This function takes a list of variables and other certain variable as parameters. If causal consistency is confirmed, the function *mem2* will return the variable and the sorted list; otherwise it will return the variable and -1. Function *mem1* is called by *mem2* and prevents the re-reading process[18].

```
fun mem1 [] (a,b,d) = false
    | mem1 ( (v1,a1,b1,d1)::xs) (a,b,d) = (a=a1
andalso b=b1 andalso d=d1)
    Orelse mem1 xs (a,b,d);

fun  mem2 [] L1 (v,a,b,d) = [(v,a,b,d)]
    |mem2 B [] (v,a,b,d) = [(v,a,b,d)]
    |mem2((v1,a1,b1,d1)::B) ((a2,b2,d2)::L1)(v,a,b,d) =
      if not(mem1 ((v1,a1,b1,d1)::B)(a2,b2,d2)) then
            [(v,a2,b2,d2)]
    else
            mem2 ((v1,a1,b1,d1)::B) L1 (v,a,b,d);
```

Functions *check* and *rm* are used for comparing two members and removing a member from the list respectively.

```
funrm a [] = []
| rm a (x::xs) = if a=x
        thenxs
            else x::(rm a xs);

fun check (x1, x2) =
        if(x1<x2) then true
        else false;
```

5. State Space Analysis

In state space analysis of the proposed model, distributed shared memory system states are developed as nodes on the state transition graph. There are two different types of deadlocks, colored Petri net deadlock and system deadlock. Colored Petri net deadlock is a model state that no transitions of the colored Petri net model are enabled, while a deadlock state of colored Petri net model may be assumed as deadlock or non-deadlock state of the system[18].

If all processes of the distributed shared memory system are finished in a last node of the state space graph, then this state is a final state. Although this state is deadlock state of the colored Petri net, it is represented that all processes are done well; and the model of the distributed shared memory system is reached to definitive state. A system state is safe so that every path that emanates from it on the state space graph may not achieve to a deadlock. Report of state space generation for the proposed model's run is according to the following:

```
---------------------------
State Space
   Nodes:  47392
   Arcs:   98507
Secs:   300
   Status: Partial

---------------------------
Scc Graph
   Nodes:  47392
   Arcs:   98507
Secs:   4
---------------------------
```

Figure 3 represents the part of the state space graph of the proposed distributed shared memory system model. This state space graph shows the path to the final state. The Petri net model achieves the final state after 98507 firings of transitions. As shown in Figure 3. All processes can run completely and conclude. Therefore, the current state of the system model is pretty safe.

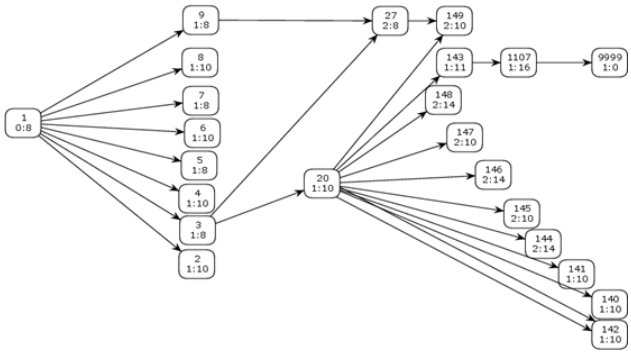

Figure 3. State space graph of the distributed shared memory system model.

6. Conclusion and Future Works

Colored Petri net is a powerful modeling language for the analysis of complex distributed systems. In this paper, a new hierarchical model of causal consistency

in distributed shared memory systems presented in full details. The results of implementations show that the proposed model takes a scenario of the performed processes in the distributed shared memory system as an input and determines whether this scenario is allowed with a causally consistent shared memory system or not, in an acceptable runtime.

7. References

1. Coulouris GF, Dollimore J, Kindberg T. Distributed systems: Concepts and design. 4th ed. New York, Harlow, England: Addison-Wesley; 2005.
2. Lamport L. How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Transactions on Computers. 1979; C-28(9):690–91.
3. Sastry JKR, Ganesh JV, Sasi Bhanu J. I^2C based networking for implementing heterogeneous microcontroller based distributed embedded systems. Indian Journal of Science and Technology. 2015; 8(15):1–10.
4. Yu H, Vahdat A. Design and evaluation of a continuous consistency model for replicated services. Proceedings of the 4th Conference on Symposium on Operating System Design and Implementation; Berkeley, USA: USENIX Association. 2000. p. 21.
5. Lamport L. On interprocess communication. Distributed Computing. 1986; 1(2):86–101.
6. Herlihy MP, Wing JM. Linearizability: A correctness condition for concurrent objects. ACM Transactions on Programming Languages and Systems (TOPLAS). 1990; 12(3):463-92.
7. Gogia R, Chhabra P, Kumari R. Consistency models in distibuted shared memory systems; 2014. p. 1–12.
8. Jensen K. Coloured petri nets, petri nets: Central models and their properties. Berlin Heidelberg : Springer; 1987. p. 248–99.
9. Huber P, Jensen K, Shapiro RM. Hierarchies in coloured petri nets. Advances in Petri Nets 1990. Berlin Heidelberg: Springer; 1991. p. 313–41.
10. Jensen K, Kristensen LM. Coloured petri nets: Modelling and validation of concurrent systems. Springer Science and Business Media; 2009.
11. Portinale L. Verification of causal models using petri nets. International Journal of Intelligent Systems. 1992; 7(8):715–42.
12. Brzezinski J, Sobaniec C, Wawrzyniak D. From session causality to causal consistency. PDP; 2004. p. 1–4.
13. Lloyd W, Freedman MJ, Kaminsky M, Andersen DG. Don't settle for eventual: Scalable causal consistency for wide-area storage with COPS. Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, ACM; New York, NY, USA. 2011. p. 401–16.
14. Ahamad M, Neiger G, Burns JE, Kohli P, Hutto PW. Causal memory: Definitions, implementation, and programming. Distributed Computing. 1995; 9(1):37–49.
15. Shen M, Kshemkalyani AD, Hsu T-Y. OPCAM: Optimal algorithms implementing causal memories in shared memory systems. Proceedings of the 2015 International Conference on Distributed Computing and Networking, ACM; New York, NY, USA. 2015. p. 1–6.
16. Paulson LC. ML for the working programmer. UK: Cambridge University Press; 1996.
17. Heiner M, Richter R, Rohr C, Schwarick M. Snoopy-A tool to design and execute graph-based formalisms. [Extended Version]. Proceedings of the 1st International Conference on Simulation tools and Techiques for Communications, Networks and Systems and Workshops; 2008. p. 15.
18. Pashazadeh S. Modeling a resource management method using hierarchical colored petri net. IEEE 1st International eConference on Computer and Knowledge Engineering (ICCKE); Mashhad, Iran. 2011. p. 34–9.