Brief Announcement: Byzantine-Tolerant Detection of Causality in Synchronous Systems

Anshuman Misra and Ajay D. Kshemkalyani(✉)

University of Illinois at Chicago, Chicago, IL 60607, USA
{amisra7,ajay}@uic.edu

Abstract. It was recently proved that the causality or the happens before relation between events in an asynchronous distributed system cannot be detected in the presence of Byzantine processes [Misra and Kshemkalyani, NCA 2022]. This result holds for the multicast, unicast, and broadcast modes of communication. This prompts us to examine whether the causality detection problem can be solved in synchronous systems in the presence of Byzantine processes. We answer this in the affirmative by outlining two approaches. The first approach uses Replicated State Machines (RSM) and vector clocks. Another approach is based on a transformation from Byzantine failures to crash failures for synchronous systems.

Keywords: Byzantine fault-tolerance · Happens before · Causality · Synchronous system · Message Passing

1 Introduction

The "happens before" or the causality relation, denoted $\rightarrow$, between events in a distributed system was defined by Lamport [6]. Given two events e and e', the *causality detection* problem asks to determine whether $e \rightarrow e'$.

There is a rich literature on solutions for solving the causality detection problem between events. See [4,5,9,15,17] for an overview of some approaches such as tracking causality graphs, scalar clocks, vector clocks [3,8], and variants of logical clocks such as hierarchical clocks, plausible clocks, dotted version vectors, Bloom clocks, interval tree clocks and resettable encoded vector clocks. Some of these variants track causality accurately while others introduce approximations as trade-offs to save on the space and/or time and/or message overheads. Schwarz and Mattern [17] stated that the quest for the holy grail of the ideal causality tracking mechanism is on. This literature above assumed that processes are correct (non-faulty). The causality detection problem for a system with Byzantine processes was recently introduced and studied in [11].

A related problem is the causal ordering of messages. Under the Byzantine failure model, causal ordering has recently been studied in [10,12,13].

S. Dolev and B. Schieber (Eds.): SSS 2023, LNCS 14310, pp. 57–61, 2023.
https://doi.org/10.1007/978-3-031-44274-2_5

Contributions. It was recently proved that the problem of detecting causality between a pair of events cannot be solved in an asynchronous system in the presence of Byzantine processes, irrespective of whether the communication is via unicasts, multicasts, or broadcasts [11]. In the multicast mode of communication, each send event sends a message to a group consisting of a subset of the set of processes in the system. Different send events can send to different subsets of processes. Communicating by unicasts and communicating by broadcasts are special cases of multicasting. It was shown in [11] that in asynchronous systems with even a single Byzantine process, the unicast and multicast modes of communication are susceptible to false positives and false negatives, whereas the broadcast mode of communication is susceptible to false negatives but no false positives. A false positive means that $e \not\rightarrow e'$ whereas $e \rightarrow e'$ is perceived/detected. A false negative means than $e \rightarrow e'$ whereas $e \not\rightarrow e'$ is perceived/detected. The impossibility result for asynchronous systems prompts us to examine whether the causality detection problem can be solved in synchronous systems in the presence of Byzantine processes. We answer in the affirmative for unicasts, multicasts, and broadcasts by outlining two approaches in this brief announcement. The results are summarized in Table 1.

Table 1. Solvability of causality detection between events under different communication modes in Byzantine-prone asynchronous and synchronous systems. FP is false positive, FN is false negative. $\overline{FP}/\overline{FN}$ means no false positive/no false negative is possible.

Mode of communication	Detecting "happens before" in asynchronous systems	Detecting "happens before" in synchronous systems
Multicasts	Impossible [11] FP, FN	Possible $\overline{FP}, \overline{FN}$
Unicasts	Impossible [11] FP, FN	Possible $\overline{FP}, \overline{FN}$
Broadcasts	Impossible [11] $\overline{FP}, FN$	Possible $\overline{FP}, \overline{FN}$

2 System Model

The distributed system is modelled as an undirected complete graph $G = (P, C)$. Here P is the set of processes communicating in the distributed system. Let $|P| = n$. C is the set of (logical) FIFO communication links over which processes communicate by message passing. The processes may be Byzantine [7,14]. The distributed system is assumed to be synchronous [2].

Let e_i^x, where $x \geq 1$, denote the x-th event executed by process p_i. An event may be an internal event, a message send event, or a message receive event. Let

the state of p_i after e_i^x be denoted s_i^x, where $x \geq 1$, and let s_i^0 be the initial state. The execution at p_i is the sequence of alternating events and resulting states, as $\langle s_i^0, e_i^1, s_i^1, e_i^2, s_i^2 \ldots \rangle$. The sequence of events $\langle e_i^1, e_i^2, \ldots \rangle$ is called the execution history at p_i and denoted E_i. Let $E = \bigcup_i \{E_i\}$ and let $T(E)$ denote the set of all events in (the set of sequences) E. The *happens before* [6] relation, denoted $\rightarrow$, is an irreflexive, asymmetric, and transitive partial order defined over events in a distributed execution that is used to define causality. The *causal past* of an event e is denoted as $CP(e)$ and defined as the set of events $\{e' \in T(E) \,|\, e' \rightarrow e\}$.

3 Problem Formulation and a Brief Overview of Solutions

The problem formulation is analogous to that in [11]. An algorithm to solve the causality detection problem collects the execution history of each process in the system and derives causal relations from it. E_i is the *actual* execution history at p_i. For any causality detection algorithm, let F_i be the execution history at p_i as perceived and collected by the algorithm and let $F = \bigcup_i \{F_i\}$. F thus denotes the execution history of the system as perceived and collected by the algorithm. Analogous to $T(E)$, let $T(F)$ denote the set of all events in F. Analogous to the definition of $\rightarrow$ on $T(E)$ [6], the *happens before* relation can be defined on $T(F)$ instead of on $T(E)$.

Let $e1 \rightarrow e2|_E$ and $e1 \rightarrow e2|_F$ be the evaluation (1 or 0) of $e1 \rightarrow e2$ using E and F, respectively. Byzantine processes may corrupt the collection of F to make it different from E. We assume that a correct process p_i needs to detect whether $e_h^x \rightarrow e_i^*$ holds and e_i^* is an event in $T(E)$. If $e_h^x \notin T(E)$ then $e_h^x \rightarrow e_i^*|_E$ evaluates to *false*. If $e_h^x \notin T(F)$ (or $e_i^* \notin T(F)$) then $e_h^x \rightarrow e_i^*|_F$ evaluates to *false*. We assume an oracle that is used for determining correctness of the causality detection algorithm; this oracle has access to E which can be any execution history such that $T(E) \supseteq CP(e_i^*)$.

Byzantine processes may collude as follows.

1. To delete e_h^x from F_h or in general, record F as any alteration of E such that $e_h^x \rightarrow e_i^*|_F = 0$, while $e_h^x \rightarrow e_i^*|_E = 1$, or
2. To add a fake event e_h^x in F_h or in general, record F as any alteration of E such that $e_h^x \rightarrow e_i^*|_F = 1$, while $e_h^x \rightarrow e_i^*|_E = 0$.

Without loss of generality, we have that $e_h^x \in T(E) \cup T(F)$. Note that e_h^x belongs to $T(F) \setminus T(E)$ when it is a fake event in F.

Definition 1. *The causality detection problem $CD(E, F, e_i^*)$ for any event $e_i^* \in T(E)$ at a correct process p_i is to devise an algorithm to collect the execution history E as F at p_i such that $valid(F) = 1$, where*

$$valid(F) = \begin{cases} 1 \text{ if } \forall e_h^x, e_h^x \rightarrow e_i^*|_E = e_h^x \rightarrow e_i^*|_F \\ 0 \text{ otherwise} \end{cases}$$

When 1 is returned, the algorithm output matches the actual (God's) truth and solves *CD* correctly. Thus, returning 1 indicates that the problem has been solved correctly by the algorithm using F. 0 is returned if either

– $\exists e_h^x$ such that $e_h^x \rightarrow e_i^*|_E = 1 \wedge e_h^x \rightarrow e_i^*|_F = 0$ (denoting a false negative), or
– $\exists e_h^x$ such that $e_h^x \rightarrow e_i^*|_E = 0 \wedge e_h^x \rightarrow e_i^*|_F = 1$ (denoting a false positive).

In our first solution, we use the Replicated State Machine (RSM) approach [16] and vector clocks in the algorithm for causality detection. We can show that F at a correct process can be made to exactly match E, hence there is no possibility of a false positive or of a false negative. The RSM approach works only in synchronous systems. In a system with n application processes, the RSM-based solution uses $3t + 1$ process replicas per application process, where t is the maximum number of Byzantine processes that can be tolerated in a RSM. Thus, there can be at most nt Byzantine processes among a total of $(3t + 1)n$ processes partitioned into n RSMs of $3t + 1$ processes each, with each RSM having up to t Byzantine processes. By using $(3t+1)n$ processes and the RSM approach to represent n application processes, the malicious effects of Byzantine process behaviors are neutralized.

Another approach is as follows. A generic transformation from Byzantine failures to crash failures for synchronous systems can be applied [1], this requires $t < n/3$. The possibility of correct Byzantine-tolerant causality detection would be implied by the possibility of correct crash-tolerant causality detection.

References

1. Bazzi, R.A., Neiger, G.: Simplifying fault-tolerance: providing the abstraction of crash failures. J. ACM **48**(3), 499–554 (2001). https://doi.org/10.1145/382780.382784
2. Dwork, C., Lynch, N.A., Stockmeyer, L.J.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988). https://doi.org/10.1145/42282.42283
3. Fidge, C.J.: Logical time in distributed computing systems. IEEE Comput. **24**(8), 28–33 (1991). https://doi.org/10.1109/2.84874
4. Kshemkalyani, A.D.: The power of logical clock abstractions. Distrib. Comput. **17**(2), 131–150 (2004). https://doi.org/10.1007/s00446-003-0105-9
5. Kshemkalyani, A.D., Shen, M., Voleti, B.: Prime clock: encoded vector clock to characterize causality in distributed systems. J. Parallel Distrib. Comput. **140**, 37–51 (2020). https://doi.org/10.1016/j.jpdc.2020.02.008
6. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7), 558–565 (1978)
7. Lamport, L., Shostak, R.E., Pease, M.C.: The Byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
8. Mattern, F.: Virtual time and global states of distributed systems. In: Parallel and Distributed Algorithms, pp. 215–226. North-Holland (1988)
9. Misra, A., Kshemkalyani, A.D.: The bloom clock for causality testing. In: Goswami, D., Hoang, T.A. (eds.) ICDCIT 2021. LNCS, vol. 12582, pp. 3–23. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-65621-8_1
10. Misra, A., Kshemkalyani, A.D.: Causal ordering in the presence of Byzantine processes. In: 28th IEEE International Conference on Parallel and Distributed Systems (ICPADS), pp. 130–138. IEEE (2022). https://doi.org/10.1109/ICPADS56603.2022.00025

11. Misra, A., Kshemkalyani, A.D.: Detecting causality in the presence of Byzantine processes: there is no holy grail. In: 21st IEEE International Symposium on Network Computing and Applications (NCA), pp. 73–80 (2022). https://doi.org/10.1109/NCA57778.2022.10013644
12. Misra, A., Kshemkalyani, A.D.: Solvability of Byzantine fault-tolerant causal ordering problems. In: Koulali, M.A., Mezini, M. (eds.) NETYS 2022. LNCS, vol. 13464, pp. 87–103. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-17436-0_7
13. Misra, A., Kshemkalyani, A.D.: Byzantine fault-tolerant causal ordering. In: 24th International Conference on Distributed Computing and Networking (ICDCN), pp. 100–109. ACM (2023). https://doi.org/10.1145/3571306.3571395
14. Pease, M.C., Shostak, R.E., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980). https://doi.org/10.1145/322186.322188
15. Pozzetti, T., Kshemkalyani, A.D.: Resettable encoded vector clock for causality analysis with an application to dynamic race detection. IEEE Trans. Parallel Distrib. Syst. **32**(4), 772–785 (2021). https://doi.org/10.1109/TPDS.2020.3032293
16. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput. Surv. **22**(4), 299–319 (1990)
17. Schwarz, R., Mattern, F.: Detecting causal relationships in distributed computations: in search of the holy grail. Distrib. Comput. **7**(3), 149–174 (1994)