algorithms

MDPI

Article

Impossibility Results for Byzantine-Tolerant State Observation, Synchronization, and Graph Computation Problems

Ajay D. Kshemkalyani [1,*] and Anshuman Misra [2]

[1] Department of Computer Science, University of Illinois Chicago, Chicago, IL 60607, USA
[2] Department of Computer Science, Purdue University Fort Wayne, Fort Wayne, IN 46805, USA; misra47@pfw.edu
* Correspondence: ajay@uic.edu

Abstract: This paper considers the solvability of several fundamental problems in asynchronous message-passing distributed systems in the presence of Byzantine processes using distributed algorithms. These problems are the following: mutual exclusion, global snapshot recording, termination detection, deadlock detection, predicate detection, causal ordering, spanning tree construction, minimum spanning tree construction, all–all shortest paths computation, and maximal independent set computation. In a distributed algorithm, each process has access only to its local variables and incident edge parameters. We show the impossibility of solving these fundamental problems by proving that they require a solution to the causality determination problem which has been shown to be unsolvable in asynchronous message-passing distributed systems.

Keywords: Byzantine fault-tolerance; "happened before" relation; causality; state observation; synchronization; graph computation; distributed system; asynchronous message-passing

1. Introduction

This paper considers the solvability of several fundamental state observation, synchronization, and graph computation problems in asynchronous message-passing distributed systems in the presence of Byzantine processes using distributed algorithms. In a *distributed algorithm*, each process has access only to its local variables and incident edge parameters (such as edge weight, edge cost, edge delay); its local variables are not accessible to any other process/node. We show the impossibility of solving these fundamental problems by proving that they require a solution to the causality determination problem [1–3], which has been shown to be unsolvable in asynchronous message-passing distributed systems [4].

In a seminal paper, Lamport formulated the "happened before" or the causality relation, denoted $\rightarrow$, between events in an asynchronous distributed system [5]. Given two events e and e', the *causality determination* (CD) problem asks to determine whether $e \rightarrow e'$. Examples of events from real-world applications include users making bids in an online auction, physical parameters like temperature and pH value reaching certain thresholds in a chemical manufacturing plant, and program variable values in a parallel program satisfying an application-specific predicate. In computing systems, applications of causality determination include determining consistent recovery points in distributed databases, deadlock detection, termination detection, distributed predicate detection, distributed debugging and monitoring, and the detection of race conditions and other synchronization errors [6]. It was shown in [4] that it is impossible to determine the causality or the happens before relation $\rightarrow$ between two events $e1$ and $e2$ when there is even a single Byzantine process in an asynchronous message-passing distributed system. False negatives

check for updates

Academic Editor: Daniele Frigioni

Received: 5 November 2024
Revised: 13 December 2024
Accepted: 30 December 2024
Published: 5 January 2025

Citation: Kshemkalyani, A.D.; Misra, A. Impossibility Results for Byzantine-Tolerant State Observation, Synchronization, and Graph Computation Problems. *Algorithms* **2025**, *18*, 26. https://doi.org/10.3390/a18010026

and/or false positives are possible. A false negative means that $e \rightarrow e'$ whereas $e \not\rightarrow e'$ is perceived/detected. A false positive means that $e \not\rightarrow e'$ whereas $e \rightarrow e'$ is detected. Specifically, the following results were shown.

1. It is impossible to determine causality $e \rightarrow e'$ between events in the presence of even a single Byzantine process when e is a communication (send or receive) event and processes communicate by unicasting. This is because both false positives and false negatives can occur.
2. A similar impossibility result when processes communicate by broadcasting was shown. In this case, false positives cannot occur but false negatives can occur.
3. A similar impossibility result to the unicasting case was shown where processes communicate by multicasting. Both false positives and false negatives can occur.

We show that many problems in distributed computing in the presence of Byzantine processes, which might be locally solved at event(s) at individual processes but require another process to detect the occurrence of such event(s), are not solvable in asynchronous message-passing systems by showing that solving them requires solving the CD problem. This also establishes the CD problem as a fundamental first-class problem as all the other problems for which we show impossibility results inherently require causality determination between a pair of events. The occurrence of false negatives (false positives) in the CD problem manifests as the occurrence of liveness and safety violations in these problems. A direct implication of our results is that none of the many algorithms proposed to solve these problems over the past five decades for failure-free systems/crash failures can be adapted for Byzantine failures.

We consider the following problems; the reader is referred to any standard textbook such as [6–10] for a centralized source of algorithms to solve these problems in asynchronous message-passing failure-free systems/graphs.

1. Synchronization and state observation problems.
   (a) Distributed mutual exclusion (ME). This problem requires enforcing that only a single process is in control (in execution) of a specific piece of code called the *critical section* at any point in time.
   (b) Global snapshot recording (GSR). This problem requires recording the states of distributed processes and communication channels in a consistent manner so that no process records an effect whose cause has not been recorded at other processes. More specifically, no message is recorded as received by a receiving process in its recorded state if the corresponding message sender process has not recorded that message as sent in its recorded state.
   (c) Termination detection (TD). This problem requires determining the occurrence of a consistent global state such that all processes have gone from an *active* to *passive* (locally terminated) state and there are no messages in transit whose receipt would cause the recipient to transition from a *passive* to *active* state.
   (d) Distributed deadlock detection (DD). When a process requests resources, it blocks on other processes. A process can block in the single-request model, OR request model, AND request model, or the AND-OR request model. Distributed deadlock detection requires determining a consistent global state in which the processes are blocked on each other in a cycle or more generally a knot topology, for the above request models.
   (e) Distributed predicate detection (PD). This problem requires detecting a consistent global state in which the variables local to different processes satisfy a predicate ϕ. The predicate may be an arbitrary relational predicate such as

$x_i + y_j > 10$ or a restricted conjunctive predicate such as $x_i = 3 \wedge y_j = 8$, where x_i and y_j are variables local to processes p_i and p_j, respectively.

(f) Causal ordering of messages (CO). This problem requires enforcing that if $s_1 \rightarrow s_2$ for send events s_1 and s_2, then for all common destinations of the corresponding messages m_1 and m_2, m_2 is not delivered before m_1.

2. Distributed graph problems.

(a) Spanning tree construction (ST). A spanning tree of a graph $G = (V, L)$ where V is the set of nodes and L is the set of edges is an acyclic sub-graph $ST = (V, L')$ where $|L'| = |V| - 1$. The ST construction problem requires identifying edges of a graph as belonging to L'.

(b) Distributed minimum spanning tree construction (MST). In a weighted graph, a MST is a spanning tree having a minimum sum of edge weights among all spanning trees of the graph. The MST construction problem requires identifying edges of a graph as belonging to the MST.

(c) All–all shortest paths construction (AASP). Given a weighted graph, this problem requires identifying the shortest length paths from each node to each other node.

(d) Maximal independent set construction (MIS). An MIS of a graph $G = (V, L)$ is a subset $V' \subseteq V$ such that for all $u, v \in V'$, $(u, v) \notin L$ and $\nexists w \in V \setminus V'$ such that $V' \cup \{w\}$ is an MIS. The MIS construction problem requires identifying the nodes in an MIS.

Solving these problems using distributed algorithms requires the determination of the existence of a causal path between two events e and e' where e is an event where a process finishes setting its local variables as a result of the distributed algorithm and e' is an event where an(other) process detects the global completion of the distributed algorithm in order to use/further process the result of the distributed algorithm. The determination of the existence of such a causal path in the execution in a Byzantine system is not solvable as shown in [4] and hence, these problems are also not solvable.

Finally, we generalize our results and show that any problem which uses a distributed algorithm is subject to at least the same limitations as the CD problem in a Byzantine failure-prone system.

The area of distributed computing is known for many impossibility results, even for the more benign crash failure model—such as for the consensus problem in asynchronous systems [11]. Or, for example, it is known that mutual exclusion cannot be solved even in a crash-prone system, so the result also applies to Byzantine failures. Lynch [12,13] has given a hundred impossibility results in distributed computing. Other impossibility results have been given in [14,15]. These impossibility results identified several classes of more basic tasks or more elementary problems that need to be solved in order to solve these problems. However, none of these more basic tasks was identified as the task of causality determination between events. In our paper, the impossibility results for the problems we identify are related to the impossibility of solving the more basic task—causality determination between events.

Previously, Lynch [12,13] observed that (in the shared memory architecture), the inherent limitations are imposed by local knowledge. This complemented Chandy-Misra's results on how processes learn [16] via message chains hints at our results which are in the context of Byzantine processes. While some of our results may not be very surprising, they nevertheless state and formalize an important outcome for a large number of important, real-world, and practical problems in asynchronous message-passing distributed systems subject to Byzantine failures that have not been previously enunciated. All these problems require relating the partial solutions of the problem at various processes to detecting at another process that these partial solutions have been reached.

Roadmap. Section 2 gives the system model. Section 3 formulates the problem of determining causality. Section 4 gives our main impossibility results about the solvability of basic problems using distributed algorithms in the Byzantine failure model. Section 5 gives a discussion and concludes.

2. System Model

We consider an asynchronous distributed system having Byzantine processes which are processes that can misbehave [17]. A correct process behaves exactly as specified by the algorithm whereas a Byzantine process may deviate arbitrarily from its protocol by exhibiting arbitrary behaviour at any point during the execution. A Byzantine process cannot impersonate another process or spawn new processes.

The distributed system is modeled as an undirected graph $G = (P, C)$. Here, P is the set of processes communicating asynchronously in the distributed system. Let $|P| = n$. C is the set of FIFO (logical) communication links over which processes communicate by message passing. A process is interchangably used with a node in the graph.

The distributed system is asynchronous, i.e., there is no fixed upper bound δ on the message latency, nor any fixed upper bound ψ on the relative speeds of processors [18].

In this paper, we consider only distributed algorithms to solve various problems. A *distributed algorithm* is one in which each process has access only to its local variables and incident edge parameters; its local variables are not accessible to any other process/node. Exchange of variable values can be carried out explicitly through message-passing. The adjacent process may be Byzantine and hence, information received from it can corrupt local variables.

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 *execution history* at p_i is the finite execution at p_i up to the current or most recent or specified local state. The *happened before* [5] relation, denoted $\rightarrow$, is an irreflexive, asymmetric, and transitive partial order defined over events in a distributed execution that is used to define causality.

Definition 1. *The happened before relation $\rightarrow$ on events consists of the following rules:*

1. ***Program Order**: For the sequence of events $\langle e_i^1, e_i^2, \ldots \rangle$ executed by process p_i, $\forall\, x, y$ such that $x < y$, we have $e_i^x \rightarrow e_i^y$.*
2. ***Message Order**: If event e_i^x is a message send event executed at process p_i and e_j^y is the corresponding message receive event at process p_j, then $e_i^x \rightarrow e_j^y$.*
3. ***Transitive Order**: If $e \rightarrow e' \wedge e' \rightarrow e''$, then $e \rightarrow e''$.*

Definition 2. *The* causal past *of an event e is denoted as $CP(e)$ and is defined as the set of events in E that causally precede e under $\rightarrow$.*

3. The Causality Determination Problem Formulation

The problem formulation in this section is based on [4]. An algorithm to solve the causality determination problem collects the execution history of each process in the system and derives causal relations from it. Let E_i denote the *actual* execution history at p_i and let $E = \bigcup_i \{E_i\}$. For any causality determination 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 as collected by the algorithm. Let $T(E)$ and $T(F)$ denote the sets of all events in E and F, respectively. Analogous to Definition 1, we can define the *happened before* relation on $T(F)$ instead of on $T(E)$.

Let $e1 \rightarrow e2|_E$ and $e1 \rightarrow e2|_F$ be the evaluation (1 (*true*) or 0 (*false*)) 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 determine 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 the correctness of the causality determination 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 3. *The causality determination 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 one is returned, the algorithm output matches the actual truth and solves *CD* correctly. Thus, returning one indicates that the problem has been solved correctly by the algorithm using F. A value of 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, abbreviated *FN*); 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, abbreviated *FP*).

To determine whether *CD* is solved correctly, we have to evaluate $\forall e_h^x, e_h^x \rightarrow e_i^*|_E = e_h^x \rightarrow e_i^*|_F$ even if $e_h^x \in T(F) \setminus T(E)$ because such an e_h^x is recorded by the algorithm as part of F. The key observation we make is that in *CD*, a single Byzantine process p_b can cause F (as recorded by the algorithm) to be different from E.

- An FN arises because a send–receive event pair (e_f^u, e_g^v) of E in a causal chain from e_h^x to e_i^* is missing as per F. In addition, an FN may arise if e_h^x is a receive event or an internal event, $e_h^x \in E \setminus F$.
- An FP arises because a non-existent send–receive message pair (e_f^u, e_g^v) in E appears in a causal chain from e_h^x to e_i^* as per F. In addition, an FP may arise if e_h^x is an internal event, $e_h^x \in F \setminus E$.

It has been proved in [4] that for send and receive events, solving the CD problem in asynchronous message-passing systems prone to Byzantine process failures is subject to both false positives and false negatives under the unicast and multicast modes of communication, and subject to false negatives under the broadcast mode of communication.

4. Impossibility Results

Consider the following class of problems. There are events $e_h^{x_h}$ at which local (possibly partial) solution(s) at p_h are obtained but which require the detection of such $e_h^{x_h}$ events at some event e_i^y at a remote process p_i. In the presence of Byzantine processes, such problems are not solvable in asynchronous message-passing systems because this requires

solving the CD problem. This also establishes the CD problem as a fundamental first-class problem as these other problems for which we show impossibility results inherently require causality determination between a pair of events. The occurrence of false negatives (false positives) in the CD problem manifests as the occurrence of liveness and safety violations in these problems. A direct implication of our results is that none of the many algorithms proposed to solve these problems over the past five decades for failure-free systems/crash failures [6–10] can be adapted for Byzantine failures.

We begin by showing the following result regarding internal events at a process.

Theorem 1. *For an internal event e_h^x, it is impossible to prevent false negatives or false positives in determining $e_h^x \rightarrow e_i^*$ correctly at a correct process p_i, i.e., matching $e_h^x \rightarrow e_i^y|_E = e_h^x \rightarrow e_i^y|_F$, in an asynchronous message passing system with one or more Byzantine processes.*

Proof. There may be no other event in the rest of the system to corroborate the occurrence of an internal event at a process. A Byzantine process p_h can choose not to reveal an internal event e_h^x to the rest of the system, leading to a false negative that cannot be prevented. It may also choose to add a fake internal event e_h^x in what it reveals to the rest of the system, leading to a false positive that cannot be prevented. □

For the problems for which we are about to show the impossibility results, the event e_h^x under consideration is seen as an internal event.

4.1. Synchronization and State Observation Problems

4.1.1. Distributed Mutual Exclusion (ME)

The ME problem is specified as follows.

- **Safety specification of ME** states that no two processes should gain access to the critical section (CS) at the same time.
- **Liveness specification of ME** states that some process should eventually be able to gain access to the critical section (CS). In addition, fairness requirements of varying degrees of stringency are typically specified.

Theorem 2. *In a system with even one Byzantine process, the distributed ME problem is subject to the same limitations (exposure to false positives and false negatives) as the CD problem, resulting in safety and liveness violations.*

Proof. Solving ME requires satisfying

$$\Phi_{ME}(e_i^{\hat{y}}) \stackrel{def}{=} (e_h^x \rightarrow e_i^y|_E = e_h^x \rightarrow e_i^y|_F = 1) \wedge \phi(e_i^{\hat{y}}),$$

where e_h^x is an "exit CS (critical section)" event, $y \leq \hat{y}$, ϕ is a predicate capturing the other requirements of the ME problem besides the first predicate, $e_i^{\hat{y}}$ is an event where the CS is entered, and $\rightarrow$ is defined on messages of the ME algorithm.

As e_h^x is an internal event, from Theorem 1 for the CD problem, detecting $e_h^x \rightarrow e_i^y$ is susceptible to false positives and false negatives. Thus, it cannot be guaranteed that the predicate $e_h^x \rightarrow e_i^y|_E = e_h^x \rightarrow e_i^y|_F$ in the formula Φ_{ME} can be satisfied. Hence, ME cannot be solved.

A false positive of the CD problem is a safety violation—multiple processes in CS—in the ME problem. A false negative of the CD problem is a liveness violation—no process can enter CS—in the ME problem. □

4.1.2. Global Snapshot Recording (GSR)

The GSR problem is specified as follows.

- **Safety specification of GSR** states that a recorded global state should include the recording of the local state of each process, that all in-transit messages in each channel should be recorded, and that such a global state should be consistent.
- **Liveness specification of GSR** states that once a recording of a global snapshot is initiated, its recording should be eventually completed.

Theorem 3. *In a system with even one Byzantine process, the distributed GSR problem is subject to the same limitations (exposure to false positives and false negatives) as the CD problem, resulting in safety and liveness violations.*

Proof. Solving GSR requires satisfying

$$\Phi_{GSR}(e_i^{\hat{y}}) \stackrel{def}{=} \bigwedge_h (e_h^{x_h} \rightarrow e_i^{y_h}|_E = e_h^{x_h} \rightarrow e_i^{y_h}|_F = 1) \wedge \phi(e_i^{\hat{y}}),$$

where $e_h^{x_h}$ is an event where the process records its local state, $(\forall h)\, y_h \leq \hat{y}$, ϕ is a predicate capturing the other requirements of the GSR problem (the recorded local states at the various processes are consistent, the channel states recording is complete) besides the first predicate, $e_i^{\hat{y}}$ is an event where the completion of the global state recording is detected, and $\rightarrow$ is defined on snapshot recording algorithm messages.

As $e_h^{x_h}$ is an internal event, from Theorem 1 for the CD problem, detecting $e_h^{x_h} \rightarrow e_i^{y_h}$ is susceptible to false positives and false negatives. Thus, it cannot be guaranteed that the predicate $e_h^{x_h} \rightarrow e_i^{y_h}|_E = e_h^{x_h} \rightarrow e_i^{y_h}|_F$ in the formula Φ_{GSR} can be satisfied. Hence, GSR cannot be solved.

A false positive of the CD problem can result in a safety violation—an inconsistent global state, supposing the false positive event is where the process is supposed to record its local state as per the algorithm but does not, receives application messages, and later records the local state—in the GSR problem. Alternately, in the definition of Φ_{GSR}, let e_h^x be an event where the process completes the recording of its local state and states of incoming channels. A false positive of the CD problem can result in a safety violation—an incomplete global state, with some local states and channel states not recorded—in the GSR problem. A false negative of the CD problem is a liveness violation—a global snapshot recording detection never occurs—in the GSR problem. □

4.1.3. Termination Detection (TD)

The TD problem is specified as follows.

- **Safety specification of TD** states global termination—all processes passive and no in-transit application messages in a (transitless) consistent global state—should not be declared unless global termination has occurred.
- **Liveness specification of TD** states that some process should eventually be able to detect global termination once it has occurred.

Theorem 4. *In a system with even one Byzantine process, the distributed TD problem is subject to the same limitations (exposure to false positives and false negatives) as the CD problem, resulting in safety and liveness violations.*

Proof. Solving TD requires satisfying

$$\Phi_{TD}(e_i^{\hat{y}}) \stackrel{def}{=} \bigwedge_h (e_h^{x_h} \rightarrow e_i^{y_h}|_E = e_h^{x_h} \rightarrow e_i^{y_h}|_F = 1) \wedge \phi(e_i^{\hat{y}}),$$

where $e_h^{x_h}$ is an event where the process becomes passive, $(\forall h)\, y_h \leq \hat{y}$, ϕ is a predicate capturing the other requirements of the TD problem—other processes are passive in a transitless consistent global state—besides the first predicate, $e_i^{\hat{y}}$ is an event where global termination is detected, and $\rightarrow$ is defined on termination detection algorithm messages.

As $e_h^{x_h}$ is an internal event, from Theorem 1 for the CD problem, detecting $e_h^{x_h} \rightarrow e_i^{y_h}$ is susceptible to false positives and false negatives. Thus, it cannot be guaranteed that the predicate $e_h^{x_h} \rightarrow e_i^{y_h}|_E = e_h^{x_h} \rightarrow e_i^{y_h}|_F$ in the formula Φ_{TD} can be satisfied. Hence, TD cannot be solved.

A false positive of the CD problem is a safety violation—no real global termination has occurred—in the TD problem. A false negative of the CD problem is a liveness violation—real global termination is not detectable—in the TD problem. □

4.1.4. Distributed Deadlock Detection (DD)

The DD problem is specified as follows.

- **Safety specification of DD** states that only a process that is part of a deadlock cycle or knot should be aborted/killed as part of deadlock resolution.
- **Liveness specification of DD** states that once a deadlock (cycle or knot in the wait-for graph) occurs in a consistent global state it should be detected and deadlock resolution performed.

Theorem 5. *In a system with even one Byzantine process, the distributed DD problem is subject to the same limitations (exposure to false positives and false negatives) as the CD problem, resulting in safety and liveness violations.*

Proof. Solving DD requires satisfying

$$\Phi_{DD}(e_i^{\hat{y}}) \stackrel{def}{=} \bigwedge_h (e_h^{x_h} \rightarrow e_i^{y_h}|_E = e_h^{x_h} \rightarrow e_i^{y_h}|_F = 1) \wedge \phi(e_i^{\hat{y}}),$$

where $e_h^{x_h}$ is an event where the process gets blocked, $(\forall h)\, y_h \leq \hat{y}$, ϕ is a predicate capturing the other requirements of the DD problem (existence of a cycle or knot in the wait-for graph in a consistent global state) besides the first predicate, $e_i^{\hat{y}}$ is an event where the deadlock is detected, and $\rightarrow$ is defined on deadlock detection algorithm messages.

As $e_h^{x_h}$ is an internal event, from Theorem 1 for the CD problem, detecting $e_h^{x_h} \rightarrow e_i^{y_h}$ is susceptible to false positives and false negatives. Thus, it cannot be guaranteed that the predicate $e_h^{x_h} \rightarrow e_i^{y_h}|_E = e_h^{x_h} \rightarrow e_i^{y_h}|_F$ in the formula Φ_{DD} can be satisfied. Hence, DD cannot be solved.

A false positive of the CD problem can result in a safety violation—unnecessary abortion—in the DD problem. A false negative of the CD problem is a liveness violation—deadlock not detectable—in the DD problem. □

4.1.5. Distributed Predicate Detection (PD)

The PD problem is specified as follows.

- **Safety specification of PD** states that a global predicate ψ is not declared as true when the global predicate is false.

- **Liveness specification of PD** states that some process should eventually be able to detect that a global predicate had become true after the predicate became true.

Theorem 6. *In a system with even one Byzantine process, the distributed PD problem is subject to the same limitations (exposure to false positives and false negatives) as the CD problem, resulting in safety and liveness violations.*

Proof. Solving PD requires satisfying

$$\Phi_{PD}(e_i^{\hat{y}}) \stackrel{def}{=} \bigwedge_h (e_h^{x_h} \rightarrow e_i^{y_h}|_E = e_h^{x_h} \rightarrow e_i^{y_h}|_F = 1) \wedge \phi(e_i^{\hat{y}}),$$

where $e_h^{x_h}$ is an event where the local variable at the process takes a value that can satisfy the global predicate ψ, $(\forall h)\, y_h \leq \hat{y}$, ϕ is a predicate capturing the other requirements of the PD problem—conditions on how the various local variable values can be combined to satisfy the global predicate ψ—besides the first predicate, $e_i^{\hat{y}}$ is an event where the global predicate ψ is detected as true, and $\rightarrow$ is defined on predicate detection algorithm messages.

As $e_h^{x_h}$ is an internal event, from Theorem 1 for the CD problem, detecting $e_h^{x_h} \rightarrow e_i^{y_h}$ is susceptible to false positives and false negatives. Thus, it cannot be guaranteed that the predicate $e_h^{x_h} \rightarrow e_i^{y_h}|_E = e_h^{x_h} \rightarrow e_i^{y_h}|_F$ in the formula Φ_{PD} can be satisfied. Hence, PD cannot be solved.

A false positive of the CD problem is a safety violation—there is no real satisfaction of the global predicate ψ—in the PD problem. A false negative of the CD problem is a liveness violation—the global predicate ψ that becomes true is never detected (because the process did not disclose its local value that could satisfy ψ)—in the PD problem. □

4.1.6. Causal Ordering of Messages (CO)

The CO problem is specified as follows [19–22].

- **Safety specification of CO** states that if the send event of message m causally happens before send event of message m', then at each common destination of m and m', m' cannot be delivered before m.
- **Liveness specification of CO** states that a message sent by a correct process to another correct process should be eventually delivered.

Theorem 7. *In a system with even one Byzantine process, the distributed CO problem is subject to the same limitations (exposure to false positives and false negatives) as the CD problem, resulting in safety and liveness violations.*

Proof. Solving CO requires satisfying

$$\Phi_{CO}(e_i^{\hat{y}}) \stackrel{def}{=} \bigwedge_{e_h^x} (e_h^x \rightarrow e_j^z|_E = e_h^x \rightarrow e_j^z|_F) \wedge \phi(e_i^{\hat{y}}),$$

where e_h^x is a send event of a message to p_i, e_j^z is an event where p_j sends a message m^* to p_i, ϕ is a predicate on when/whether p_i can safely deliver m^* sent at e_j^z to itself (i.e., has received and determines it is safe to give m^* with respect to all other messages sent to itself in the execution to the application), $e_i^{\hat{y}}$ is an event where p_i delivers the message m^* from p_j, and $\rightarrow$ is defined on application messages.

As e_h^x is a send event, from [4] for the CD problem, detecting $e_h^x \rightarrow e_j^z$ is susceptible to false positives and/or false negatives. Thus, it cannot be guaranteed that the predicate $e_h^x \rightarrow e_j^z|_E = e_h^x \rightarrow e_j^z|_F$ in the formula Φ_{CO} can be satisfied. Hence, CO cannot be solved.

A false positive of the CD problem can result in a liveness violation—waiting indefinitely at p_i for the delivery of m^* until the prior delivery of m' that was never sent by p_h—in the CO problem. A false negative of the CD problem is a safety violation—not waiting for the delivery of m' that was sent by p_h at e_h^x to p_i—in the CO problem. □

4.2. Distributed Graph Problems

4.2.1. Spanning Tree Construction (ST)

The ST problem is specified as follows.

- **Safety specification of ST** states that a spanning tree (having $n-1$ edges and an acyclic sub-graph) is selected.
- **Liveness specification of ST** states that some process should eventually be able to detect that the spanning tree construction is completed.

Theorem 8. *In a system with even one Byzantine process, the distributed ST construction problem is subject to the same limitations (exposure to false positives and false negatives) as the CD problem, resulting in safety and liveness violations.*

Proof. Solving ST requires satisfying

$$\Phi_{ST}(e_i^{\hat{y}}) \stackrel{def}{=} \bigwedge_h (e_h^{x_h} \rightarrow e_i^{y_h}|_E = e_h^{x_h} \rightarrow e_i^{y_h}|_F = 1),$$

where $e_h^{x_h}$ is an event where p_h has selected its incident spanning tree edges (of an actual spanning tree), $(\forall h)\, y_h \leq \hat{y}$, $e_i^{\hat{y}}$ is an event where p_i determines that the distributed spanning tree determination is complete, and $\rightarrow$ is defined on spanning tree algorithm messages.

As $e_h^{x_h}$ is an internal event, from Theorem 1 for the CD problem, detecting $e_h^{x_h} \rightarrow e_i^{y_h}$ is susceptible to false positives and false negatives. Thus, it cannot be guaranteed that the predicate $e_h^{x_h} \rightarrow e_i^{y_h}|_E = e_h^{x_h} \rightarrow e_i^{y_h}|_F$ in the formula Φ_{ST} can be satisfied. Hence, ST cannot be solved.

A false positive of the CD problem is a safety violation—a cycle or a non-tree sub-graph is created instead of a spanning tree—in the ST problem. A false negative of the CD problem is a liveness violation—completion of the distributed spanning tree construction is not detectable to any p_i—in the ST problem. □

4.2.2. Minimum Spanning Tree Construction (MST)

The MST problem is specified as follows.

- **Safety specification of MST** states that a spanning tree (having $n-1$ edges and an acyclic sub-graph) having the minimum possible sum of edge weights is selected.
- **Liveness specification of MST** states that some process should eventually be able to detect that the minimum spanning tree construction is completed.

Theorem 9. *In a system with even one Byzantine process, the distributed MST construction problem is subject to the same limitations (exposure to false positives and false negatives) as the CD problem, resulting in safety and liveness violations.*

Proof. Solving MST requires satisfying

$$\Phi_{MST}(e_i^{\hat{y}}) \stackrel{def}{=} \bigwedge_h (e_h^{x_h} \rightarrow e_i^{y_h}|_E = e_h^{x_h} \rightarrow e_i^{y_h}|_F = 1),$$

where $e_h^{x_h}$ is an event where p_h has selected its incident spanning tree edges (of an actual minimum spanning tree), $(\forall h)\, y_h \leq \hat{y}$, $e_i^{\hat{y}}$ is an event where p_i determines that the distributed determination of the minimum spanning tree is complete, and $\rightarrow$ is defined on minimum spanning tree algorithm messages.

As $e_h^{x_h}$ is an internal event, from Theorem 1 for the CD problem, detecting $e_h^{x_h} \rightarrow e_i^{y_h}$ is susceptible to false positives and false negatives. Thus, it cannot be guaranteed that the predicate $e_h^{x_h} \rightarrow e_i^{y_h}|_E = e_h^{x_h} \rightarrow e_i^{y_h}|_F$ in the formula Φ_{MST} can be satisfied. Hence, MST cannot be solved.

A false positive of the CD problem is a safety violation—a cyclic sub-graph or a non-tree sub-graph or a non-minimal spanning tree is identified as a minimum spanning tree—in the MST problem. A false negative of the CD problem is a liveness violation—completion of the minimum spanning tree construction is not detectable by any p_i—in the MST problem. □

4.2.3. All–All Shortest Paths Construction (AASP)

The AASP problem is specified as follows.

- **Safety specification of AASP** states that for each node of a graph acting as a source (or sink) node, (the spanning tree representing) the shortest paths to (or from) every other node are selected.
- **Liveness specification of AASP** states that each process should eventually be able to detect that the construction of the shortest paths spanning tree rooted at itself is completed.

Theorem 10. *In a system with even one Byzantine process, the distributed AASP construction problem is subject to the same limitations (exposure to false positives and false negatives) as the CD problem, resulting in safety and liveness violations.*

Proof. Solving AASP requires satisfying

$$(\forall i)\Phi_{AASP}(e_i^{\hat{y}^i}) \stackrel{def}{=} \bigwedge_i (\bigwedge_h (e_h^{x_h^i} \rightarrow e_i^{y_h^i}|_E = e_h^{x_h^i} \rightarrow e_i^{y_h^i}|_F = 1)),$$

where $e_h^{x_h^i}$ is an event where p_h has identified its adjacent edges in a shortest paths sink tree rooted at p_i (of an actual shortest path sink tree of p_i), $(\forall h)\, y_h^i \leq \hat{y}^i$, $e_i^{\hat{y}^i}$ is an event where p_i determines that the distributed determination of the shortest path sink tree rooted at itself is complete, and $\rightarrow$ is defined on the shortest path sink tree algorithm messages.

As $e_h^{x_h^i}$ is an internal event, from Theorem 1 for the CD problem, detecting $e_h^{x_h^i} \rightarrow e_i^{y_h^i}$ is susceptible to false positives and false negatives. Thus, it cannot be guaranteed that for any i, the predicate $e_h^{x_h^i} \rightarrow e_i^{y_h^i}|_E = e_h^{x_h^i} \rightarrow e_i^{y_h^i}|_F$ in the formula Φ_{AASP} can be satisfied. Hence, AASP cannot be solved.

A false positive of the CD problem is a safety violation—a shortest paths sink tree is not used as the sink tree rooted at some p_i—in the AASP problem. A false negative of the CD problem is a liveness violation—completion of the construction of the shortest paths sink tree rooted at some p_i is not detectable by that p_i—in the AASP problem. □

4.2.4. Maximal Independent Set Construction (MIS)

The MIS problem is specified as follows.

- **Safety specification of MIS** states that no two nodes that are neighbors add themselves to the maximal independent set and no superset of the set so constructed satisfies the independent set property.

- **Liveness specification of MIS** states that some process should eventually be able to detect that the maximal independent set construction is complete.

Theorem 11. *In a system with even one Byzantine process, the distributed MIS construction problem is subject to the same limitations (exposure to false positives and false negatives) as the CD problem, resulting in safety and liveness violations.*

Proof. Solving MIS requires satisfying

$$\Phi_{MIS}(e_i^{\hat{y}}) \stackrel{def}{=} \bigwedge_h (e_h^{x_h} \rightarrow e_i^{y_h}|_E = e_h^{x_h} \rightarrow e_i^{y_h}|_F = 1),$$

where $e_h^{x_h}$ is an event where p_h has determined whether or not it belongs to the maximal independent set (in a true maximal independent set), $(\forall h)\, y_h \leq \hat{y}$, $e_i^{\hat{y}}$ is an event where p_i determines that the distributed maximal independent set construction is complete, and $\rightarrow$ is defined on maximal independent set algorithm messages.

As $e_h^{x_h}$ is an internal event, from Theorem 1 for the CD problem, detecting $e_h^{x_h} \rightarrow e_i^{y_h}$ is susceptible to false positives and false negatives. Thus, it cannot be guaranteed that the predicate $e_h^{x_h} \rightarrow e_i^{y_h}|_E = e_h^{x_h} \rightarrow e_i^{y_h}|_F$ in the formula Φ_{MIS} can be satisfied. Hence, MIS cannot be solved.

A false positive of the CD problem is a safety violation—two neighboring nodes add themselves to the maximal independent set or some node that has not added itself to the maximal independent set can be added to the maximal independent set—in the MIS problem. A false negative of the CD problem is a liveness violation—completion of the maximal independent set construction is not detectable by any p_i—in the MIS problem. □

4.3. Generalized Theorem

A distributed algorithm is an algorithm in which each process is initialized with its local variable values and incident edge parameters; no process has access to any other variables and parameters of the system. The process can communicate only with its neighboring processes (depending on the overlay, if any) along incident edges.

For any problem X which requires a distributed algorithm to solve it, there are two characteristic events. $e_h^{x_h}$ is an internal event at which process p_h completes its calculation of local variable values required to solve the problem after communicating with other processes in a distributed manner. $e_i^{\hat{y}}$ is an event at which process p_i determines that a global solution to the problem has been attained. When $h \neq i$, in order that p_i at $e_i^{\hat{y}}$ can detect that problem X has been solved, there needs to be an actual causal path from $e_h^{x_h}$ to $e_i^{y_h}$ ($\forall h$ and where $y_h \leq \hat{y}$) that is also detectable by p_i, i.e., $\bigwedge_h e_h^{x_h} \rightarrow e_i^{y_h}|_E = e_h^{x_h} \rightarrow e_i^{y_h}|_F = 1$.

- **Safety specification of** X states the correctness conditions of a solution to X, as captured by a global formula Φ_X.
- **Liveness (or termination) specification of** X states that some process should eventually be able to detect that a global formula Φ_X has become true.

Theorem 12. *In a system with even one Byzantine process, when a process p_i has to detect that a problem X has been locally solved at events $e_h^{x_h}$, the distributed X problem is subject to the same limitations (exposure to false positives and false negatives) as the CD problem, resulting in safety and liveness violations.*

Proof. Solving X requires satisfying

$$\Phi_X(e_i^{\hat{y}}) \stackrel{def}{=} \bigwedge_h (e_h^{x_h} \rightarrow e_i^{y_h}|_E = e_h^{x_h} \rightarrow e_i^{y_h}|_F = 1) \wedge \phi(e_i^{\hat{y}}),$$

where $e_h^{x_h}$ is an event where the local variables at the process take values that specify that the local computation has completed at p_h, $(\forall h)\, y_h \leq \hat{y}$, ϕ is a predicate capturing the other requirements of the X problem besides the first predicate, $e_i^{\hat{y}}$ is an event where the global formula Φ_X is detected as true, and $\rightarrow$ is defined on algorithm messages for solving X.

As $e_h^{x_h}$ is an internal event, from Theorem 1 for the CD problem, detecting $e_h^{x_h} \rightarrow e_i^{y_h}$ is susceptible to false positives and false negatives. Thus, it cannot be guaranteed that the predicate $e_h^{x_h} \rightarrow e_i^{y_h}|_E = e_h^{x_h} \rightarrow e_i^{y_h}|_F$ in the formula Φ_X can be satisfied. Hence, X cannot be solved.

A false positive of the CD problem is a safety violation—there is no real satisfaction of the global formula Φ_X—in the X problem. A false negative of the CD problem is a liveness violation—the global formula Φ_X that becomes true is never detected (because the process did not disclose its local value that could satisfy Φ_X)—in the X problem. □

5. Discussion and Conclusions

We proved the impossibility of solving ten important problems in distributed computing in an asynchronous message-passing system susceptible to Byzantine failures. The proofs were generalized to prove Theorem 12, which states the impossibility of solving any problem using a distributed algorithm that requires knowledge of a local action at a process being used by a remote event before the results of the distributed algorithm can be used. These problems require solving the causality determination (CD) problem, which has been shown to be unsolvable in such systems [4]. This also establishes the CD problem as a fundamental first-class problem, akin to the consensus problem.

By theoretically establishing these impossibility results, this paper's practical contribution is that other approaches besides such fully distributed algorithms should be used in Byzantine environments to solve the various problems identified.

Author Contributions: Conceptualization, A.D.K. and A.M.; methodology, A.D.K.; formal analysis, A.D.K.; investigation, A.D.K.; writing—original draft preparation, A.D.K.; writing—review and editing, A.D.K. and A.M.; supervision, A.D.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: No data was used or created in this research.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Schwarz, R.; Mattern, F. Detecting Causal Relationships in Distributed Computations: In Search of the Holy Grail. *Distrib. Comput.* **1994**, *7*, 149–174. [CrossRef]
2. Mattern, F. Virtual Time and Global States of Distributed Systems. In Proceedings of the Parallel and Distributed Algorithms, North-Holland, The Netherlands, October 1988; pp. 215–226.
3. Fidge, C.J. Logical Time in Distributed Computing Systems. *IEEE Comput.* **1991**, *24*, 28–33. [CrossRef]
4. Misra, A.; Kshemkalyani, A.D. Detecting Causality in the Presence of Byzantine Processes: There is No Holy Grail. In Proceedings of the 21st IEEE International Symposium on Network Computing and Applications (NCA), Boston, MA, USA, 14–16 December 2022; pp. 73–80. [CrossRef]
5. Lamport, L. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM* **1978**, *21*, 558–565. [CrossRef]
6. Kshemkalyani, A.D.; Singhal, M. *Distributed Computing: Principles, Algorithms, and Systems*; Cambridge University Press: Cambridge, UK, 2011. [CrossRef]

7. Garg, V.K. *Elements of Distributed Computing*; Wiley: Hoboken, NJ, USA, 2002.
8. Raynal, M. *Distributed Algorithms for Message-Passing Systems*; Springer: Berlin/Heidelberg, Germany, 2013. [CrossRef]
9. Tanenbaum, A.S.; van Steen, M. *Distributed Systems—Principles and Paradigms*, 2nd ed.; Pearson Education: London, UK, 2007.
10. Coulouris, G.; Dollimore, J.; Kindberg, T. *Distributed Systems—Concepts and Designs*, 3rd ed.; International Computer Science Series; Addison-Wesley-Longman: North York, ON, Canada, 2002.
11. Fischer, M.J.; Lynch, N.A.; Paterson, M.S. Impossibility of distributed consensus with one faulty process. *J. ACM (JACM)* **1985**, *32*, 374–382. [CrossRef]
12. Lynch, N. A Hundred Impossibility Results for Distributed Computing. In *MIT Technical Report MIT/LCS/TM/394*; Laboratory for Computer Science, Massachusetts Institute of Technology: Cambridge, MA, USA, 1989.
13. Lynch, N.A. A Hundred Impossibility Proofs for Distributed Computing. In Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, Edmonton, AB, Canada, 14–16 August 1989; pp. 1–28. [CrossRef]
14. Attiya, H.; Ellen, F. *Impossibility Results for Distributed Computing*; Synthesis Lectures on Distributed Computing Theory; Morgan & Claypool Publishers: San Rafael, CA, USA, 2014. [CrossRef]
15. Fich, F.E.; Ruppert, E. Hundreds of impossibility results for distributed computing. *Distrib. Comput.* **2003**, *16*, 121–163. [CrossRef]
16. Chandy, K.M.; Misra, J. How Processes Learn. *Distrib. Comput.* **1986**, *1*, 40–52. [CrossRef]
17. Lamport, L.; Shostak, R.E.; Pease, M.C. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.* **1982**, *4*, 382–401. [CrossRef]
18. Dwork, C.; Lynch, N.A.; Stockmeyer, L.J. Consensus in the presence of partial synchrony. *J. ACM* **1988**, *35*, 288–323. [CrossRef]
19. Misra, A.; Kshemkalyani, A.D. Solvability of Byzantine Fault-Tolerant Causal Ordering Problems. In Proceedings of the Networked Systems—10th International Conference, NETYS 2022, Virtual Event, 17–19 May 2022; LNCS; Springer: Cham, Switzerland, 2022; Volume 13464, pp. 87–103. [CrossRef]
20. Misra, A.; Kshemkalyani, A.D. Causal Ordering in the Presence of Byzantine Processes. In Proceedings of the 28th IEEE International Conference on Parallel and Distributed Systems ICPADS, Nanjing, China, 10–12 January 2022; pp. 130–138. [CrossRef]
21. Misra, A.; Kshemkalyani, A.D. Byzantine Fault-Tolerant Causal Ordering. In Proceedings of the 24th International Conference on Distributed Computing and Networking, ICDCN 2023, Kharagpur, India, 4–7 January 2023; pp. 100–109. [CrossRef]
22. Misra, A.; Kshemkalyani, A.D. Byzantine-Tolerant Causal Ordering for Unicasts, Multicasts, and Broadcasts. *IEEE Trans. Parallel Distrib. Syst.* **2024**, *35*, 814–828. [CrossRef]