Happened Before is the Wrong Model for Potential Causality

Ashis Tarafdar and Vijay K. Garg

TR-PDS-1998-006 July 1998

Parallel & Distributed Systems group
Department of Electrical & Computer Engineering
University of Texas at Austin
Austin, Texas 78712

Happened Before is the Wrong Model for Potential Causality *

Ashis Tarafdar
Dept. of Computer Sciences
University of Texas at Austin
Austin, TX 78712
(ashis@cs.utexas.edu)

Vijay K. Garg
Dept. of Electrical and Computer Engg.
University of Texas at Austin
Austin, TX 78712
(garg@ece.utexas.edu)

July 15, 1998

Abstract

The happened before model has been widely used to model distributed computations. In particular, it has been used to model the logical time and the potential causality aspects of a distributed computation. Though it is a good model for logical time, we argue that it is not a good model for potential causality. We introduce a better model for potential causality that extends happened before to allow independent local events to be partially ordered. This potential causality model has marked advantages over the happened before model in applications areas such as debugging and recovery.

1 Introduction

The *happened before model* [Lam78] has been used to model distributed computations, capturing the notions of *logical time* and *potential causality*. As a result, these two notions have long been considered the same. We argue that these two notions are different and, in fact, starkly contradictory in nature. They arise in different applications and require different models. Although happened before suffices to model logical time, it is not good for modeling potential causality.

The goal of this discussion is to introduce a new model of distributed computations and to motivate its use in applications that involve potential causality. The new model captures potential causality by allowing *independent* (as opposed to *concurrent*) events within the same process to be partially ordered. We demonstrate how the new model proves effective in many application areas.

2 Historical Perspective

The history of modeling distributed computations may be divided into three stages. The first stage started with Lamport's introduction of happened before to model logical time [Lam78]. The second started with Mattern's observation that happened before may also be used to model potential causality [Mat89]. The third stage consisted of the gradual discovery that using happened before to model potential causality leads to problems owing to inherent false causality [CS93, SBN+97, TG98].

*supported in part by the NSF ECS-9414780, CCR-9520540, a General Motors Fellowship, Texas Education Board ARP-320 and an IBM grant

Stage 1: Happened Before and Logical Time

Many applications, such as mutual exclusion and deterministic replay, need to know the order in which events [1] happen in time. In a distributed system, events on different processes do not share a common clock. This makes it impossible to determine their order in real time using time-stamping mechanisms. Lamport [Lam78] introduced logical time to order distributed events in a manner that approximates their real time order.

To model logical time, Lamport defined the *happened before* relation, denoted by $\rightarrow$, as the smallest relation satisfying the following: (1) If a and b are events in the same process, and a comes before b according to the total ordering specified by the local clock on that process, then $a \rightarrow b$. (2) If a is the send event of a message and b is the receipt of the same message by another process, then $a \rightarrow b$. (3) If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$. Further, two events a and b are *concurrent*, denoted by $a \| b$, if and only if they are incomparable using the happened before relation (i.e. $(a \not\rightarrow b) \wedge (b \not\rightarrow a)$).

Lamport went on to define a logical clock mechanism which constructed a total order of events that is a linearization of the happened before relation. This total order may be viewed as a possible ordering of events in real time and is sufficient for applications that need a notion of logical time.

Stage 2: Happened Before and Potential Causality

If an event happens before another event, it has the potential for causing that event. Many applications, such as recovery and debugging, require the tracking of such causal dependencies. Mattern [Mat89] realized that such applications would benefit by a mechanism to quickly determine the happened before relation between events.

The totally ordered logical clock mechanism that proved useful for applications requiring a notion of logical time is not good for applications requiring the notion of potential causality. In Mattern's own words: "For some applications (like mutual exclusion as described by Lamport himself in [Lam78]) this defect is not noticeable. For other purposes (e.g., distributed debugging), however, this is an important defect." Mattern, therefore, proposed a vector clock mechanism that allows the happened before relation between events to be deduced.

Stage 3: False Causality Problems

An event that happens before another event need not necessarily cause it. This is implicit when we say that happened before tracks *potential* causality. Therefore, an inherent problem in using happened before in applications that require causality tracking is that sometimes events that are independent are believed to have a causal dependency. This phenomenon is called *false causality*. While any approximation of causality must have false causality, the happened before model was found to fall particularly short in this respect. We cite three examples of application domains where this has happened.

Firstly, happened before has been used as the basis of causally and totally ordered communication support. Cheriton and Skeen [CS93] observed that when two send events have a false causal dependency between them, the resulting effect is to make the receipt of one message unnecessarily wait for the receipt of the other. This overhead was mentioned as one of the limitations of causally and totally ordered communication support.

[1] What exactly an *event* is would depend on the application. For example, it could be the execution of a machine instruction or the execution of a procedure. However, we do place one restriction on the level of granularity of an event – we assume that events are chosen so that any ordering under consideration is irreflexive.

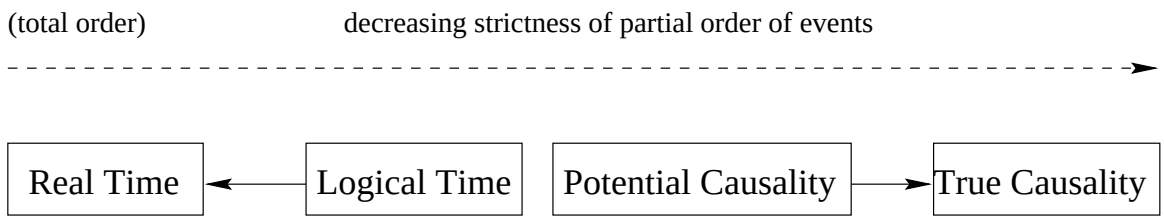


Figure 1: Spectrum of models

Secondly, happened before has been used in tools to detect data races in multi-threaded programs. Savage, et al [SBN+97] pointed out that false causality between events causes some data races to go undetected. This is because if one event happens before another, it may falsely be believed to cause that event and thus a potential data race between the independent events may be missed.

The third application domain is in predicate detection. In a previous study [TG98], we noted that using the happened before model would miss the detection of certain predicates. The reasons for this are similar to those in race detection. Since predicate detection has applications in distributed debugging, this translates to certain bugs being missed.

3 Logical Time is *not* Potential Causality

We have seen that happened before has come to be used to model both logical time and potential causality in distributed computations. This has led to a wide-spread belief that the two concepts are the same. We now argue that the concepts are not only different but opposite in nature.

The applications that motivated Lamport to model logical time were notably different from those that motivated Mattern to model potential causality. The first kind of applications, represented by mutual exclusion, require as close an approximation to real time order as possible. Happened before is, in fact, the *best* possible approximation of real time order that one can make in a message-passing distributed system without external channels.

The kind of applications that require potential causality, represented by distributed recovery, require as close an approximation as possible to the true causality between events. This true causality is captured by the following *true causes* relation between events:

Definition 1
An event a truly causes *an event b (denoted by* $a \hookrightarrow b$*) if and only if: (1) a and b conflict (i.e. the computational effect of their execution depends on the order in which they are executed), and (2) a occurs before b in real time. Further, two events a and b are* independent *(denoted by* $a|b$*) if and only if they are incomparable by the truly causes relation (i.e.* $(a \not\hookrightarrow b) \wedge (b \not\hookrightarrow a)$*).*

Thus, a send event truly causes the receive event of the same message. *However, an event that precedes another event according to the local clock of a process does not necessarily truly cause that event.* (We will elaborate on this key idea in the next section.) This is at the root of all false causality inherent in the happened before model.

As illustrated in Figure 1, real time and true causality fall at opposite ends of a spectrum of partial orders based on their strictness. While logical time tries to approximate the real time total order that lies on one end of the spectrum, potential causality tries to approximate true causality at the other end. Happened before was initially introduced to model logical time and therefore closely

Real Time	Logical Time	Potential Causality	True Causality
Occurred Before	Happened Before ($\rightarrow$)	Potentially Causes ($\overset{p}{\hookrightarrow}$)	Truly Causes ($\hookrightarrow$)
is Simultaneous with	is Concurrent with ($\parallel$)	is Potentially Independent of ($\mid_p$)	is Independent of ($\mid$)

Table 1: A Summary of Models

approximates real time. By a historical accident, it was also used to model potential causality. In fact, in attempting to approach real time, the happened before model creates more false causality than necessary. A better model for potential causality would be a less strict partial order that approaches true causality in the spectrum indicated in Figure 1.

4 The Potential Causality Model

In order to model potential causality, we must track causal dependencies. In a message-passing system without external channels, causality between processes may only be through messages. Within the same process, however, we must assume the existence of a local causality tracking mechanism (just as happened before assumed a local time tracking mechanism). We require that the local causality tracking mechanism be pessimistic: it must order local events stricter than their true causal ordering. Thus, if one local event truly causes another, it must also potentially cause the other.

Definition 2
The potentially causes *relation, denoted by* $\overset{p}{\hookrightarrow}$*, is the smallest relation satisfying the following: (1) If a and b are events in the same process, and a can be determined to potentially cause b by a local causality tracking mechanism, then* $a \overset{p}{\hookrightarrow} b$*. (2) If a is the send event of a message and b is the receipt of the same message by another process, then* $a \overset{p}{\hookrightarrow} b$*. (3) If* $a \overset{p}{\hookrightarrow} b$ *and* $b \overset{p}{\hookrightarrow} c$*, then* $a \overset{p}{\hookrightarrow} c$*. Further, two events a and b are* potentially independent*, denoted by* $a \mid_p b$*, if and only if they are incomparable by the potentially causes relation (i.e.* $(a \overset{p}{\not\rightarrow} b) \wedge (b \overset{p}{\not\rightarrow} a)$*).*

Table 1 summarizes the different models of distributed computations. Note that the ordering of events becomes less strict from left to right. In particular, for any two events a and b:

$$\begin{array}{ccccc} a \rightarrow b & \Leftarrow & a \overset{p}{\hookrightarrow} b & \Leftarrow & a \hookrightarrow b \\ a \parallel b & \Rightarrow & a \mid_p b & \Rightarrow & a \mid b \end{array}$$

Local Causality Tracking

The crucial difference between the potential causality model and the happened before model is in the local ordering of events in a process. While happened before totally orders local events in real time, potential causality partially orders local events as indicated by a causality tracking mechanism.

As long as local causality tracking is a valid approximation of true causality as defined above, we allow any partial order of local events to be a valid local ordering of events. The choice of a local causality tracking mechanism would depend on the intended application domain. The reason

for this is two-fold. Firstly, the definition of true causality has different implications in different application domains. Secondly, the application domain would determine the ease of implementation, and consequently, the closeness of approximation to true causality that is possible. We now discuss these issues.

First, we return to the key observation that we made in the previous section to distinguish between logical time and potential causality: an event that precedes another event according to the local clock of a process does not necessarily truly cause that event. This is clear in the following application domains:

• A single shared communication subsystem can receive calls from many independent applications. Events across calls from independent applications would be independent but would still be totally ordered by a local clock.

• Database systems use a model in which only the order of *conflicting* operations is important. All other operations are, in fact, independent. This allows flexibility for better concurrency and recoverability. In such a model, the local clock ordering of events is important only for those operations that conflict.

• During recovery in distributed systems, so long as the final state recovered is the same, the exact order in which events are replayed is unimportant. Events which could happen in either order with the same effect are independent regardless of their local clock ordering.

• Applications consisting of multiple processes or multiple threads have independent events across processes and threads which are ordered in an arbitrary way by the local clock. This is an example where local concurrency is a form of independence. It has often been modeled by extending the happened before model to have a separate (happened before) process for each (operating system) process or thread. Note that this is not in-line with the original intent of happened before, that of tracking real time order, and is more in-line with our causality tracking approach.

These example applications suffice to demonstrate that a local clock ordering does not correspond to true causality in a number of application domains. They also demonstrate how the definition of true causality would apply to various application domains.

The issue of implementing a local causality tracking mechanism would also be application specific. Each application must determine whether a given pair of events are potentially independent or not. A discussion of such implementation techniques, though important, is beyond the scope of this discussion.

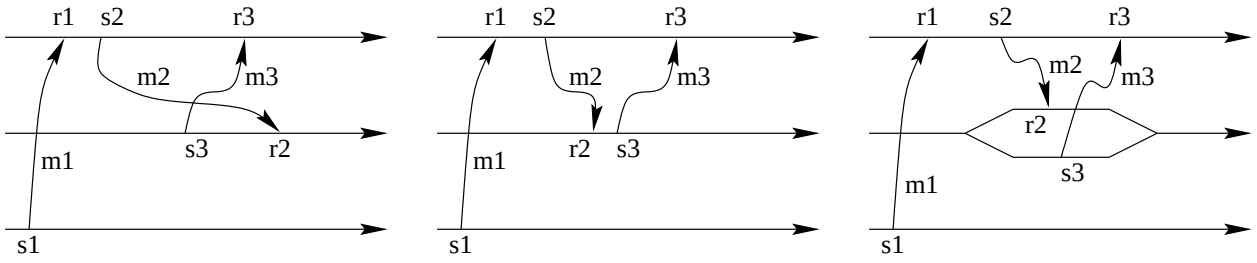
5 Applications

We have mentioned that there are a number of applications that require causality to be tracked. We now present some representative applications and demonstrate how they benefit from the use of the potential causality model. Each of the applications has been researched extensively in the literature using happened before. In each case, we show that potential causality is the more appropriate model.

5.1 Race Detection

The main reason that parallel programs are difficult to debug and test is that message races cause non-determinism. A large body of research has been devoted to detecting such races. We focus on one such work [NBDK96].

The authors use the happened before model and local events are totally ordered. A race is defined to occur when the following conditions exist: a receive operation r is pending, two or more



(a) race detected in happened before (b) no race in happened before (c) race detected in strong causality

Figure 2: Race Detection

messages are simultaneously in transit, and the messages are sent on channels over which r listens. In Figure 2(a) a race is detected because the two messages $m1$ and $m3$ may be simultaneously in transit on channels over which pending receive $r1$ is listening. On the other hand, in Figure 2(b) no race is detected because $m1$ and $m3$ cannot be simultaneously in transit. This is because the receive event $r1$ of message $m1$ happened before the send event $s3$ for $m3$. Therefore, the message $m1$ is believed to cause message $m3$.

However, the flaw in this reasoning is that $r1$ happened before $s3$ merely implies that $m1$ *may* have caused $m3$. If $r2$ and $s3$ are independent, then $m1$ and $m3$ may be in transit simultaneously, and we have a race. This is clear from the potential causality representation shown in Figure 2(c).

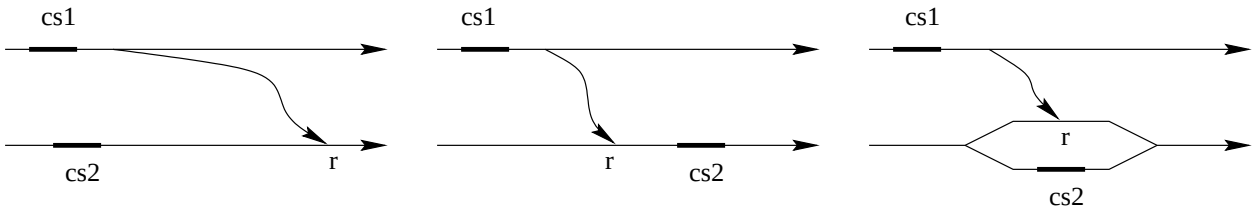
To summarize: the potential causality model allows the detection of races that would be missed in the happened before model owing to false causality.

5.2 Predicate Detection

The predicate detection problem involves detecting a condition that is defined on the combined state of multiple processes. It is a fundamental problem that has applications in many areas, such as in distributed debugging. Predicate detection has been widely studied and we select one work [GW94].

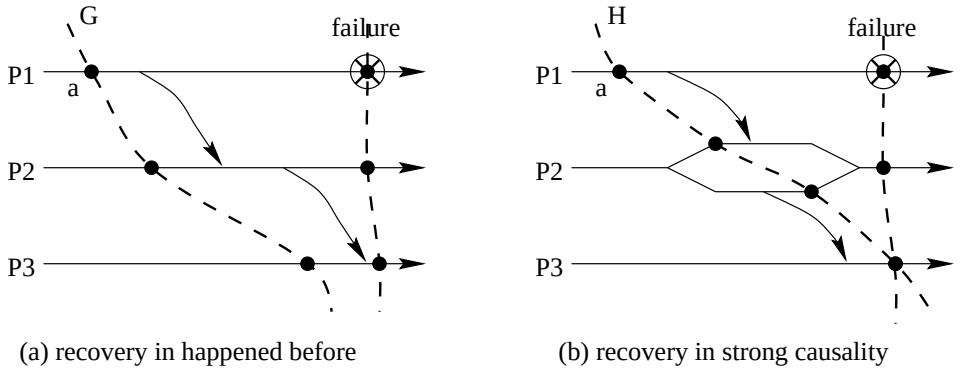
The authors define the predicate detection problem in the context of the happened before model with totally ordered local events. As an example, consider a two process system in which we would like to detect if mutual exclusion is violated. The predicate to be detected is: $cs1 \wedge cs2$ where $cs1$ and $cs2$ are local predicates that are true within a critical section. The authors give an efficient algorithm for detecting such a predicate.

In the computation shown in Figure 3(a), for example, the predicate would be detected because



(a) predicate detected in happened before (b) predicate not detected in happened before (c) predicate detected in strong causality

Figure 3: Predicate Detection



(a) recovery in happened before

(b) recovery in strong causality

Figure 4: Recovery

both critical sections may be entered simultaneously. However, in Figure 3(b), the predicate is not detected because the message makes the critical section for $cs1$ happen before the critical section for $cs2$.

However, the message may have been fortuitous and may have had nothing to do with the mutual exclusion protocol. The message induced false causality which led us to miss a bug in the execution. If we had used the potential causality model instead, the predicate could be detected as shown in Figure 3(c). This is a more accurate representation of the causal dependencies between events in the execution. In one of our recent studies [TG98], we presented algorithms for solving predicate detection in the potential causality model.

To summarize: the potential causality model allows the detection of predicates (bugs) that would be missed in the happened before model owing to false causality.

5.3 Recovery

We choose [EJW96] as a recent survey of rollback-recovery protocols in message-passing systems. The authors classify rollback-recovery into two categories: checkpoint-based rollback-recovery and log-based rollback-recovery. To simplify our presentation, we limit our focus to log-based rollback-recovery. Our arguments can be extended to cover checkpoint-based rollback-recovery as well.

In Figure 4(a), process $P1$ has failed and has recovered to state a (using checkpoints and message logs). In order to maintain consistency, any events that causally depend on the events on $P1$ that have been rolled back must also be rolled back. The most recent consistent global state to which the whole system can roll back is termed the *maximum recoverable state*, (G in the figure).

As expected of the happened before model, some events may be unnecessarily rolled back because of false causality. If the receive and send events on process $P2$ were independent as in Figure 4(b), then the *maximum recoverable state* can be advanced (to H) to avoid unnecessary rollbacks. In this case, process $P2$ only needs to roll back its receive event and process $P3$ doesn't need to roll back at all.

To summarize: the potential causality model requires less rollback during recovery than the happened before model, and thus results in less wasted work.

As an aside, we note that log-based recovery schemes are classified into pessimistic, optimistic, and causal schemes. Reducing the extent of roll-back has an impact on optimistic schemes. Using potential causality also has benefits for causal log-based recovery [EZ92, AHM93]. These schemes explicitly track causality using happened before. The potential causality model would identify false causality, and thus reduce the number of piggy-backed messages and message logs.

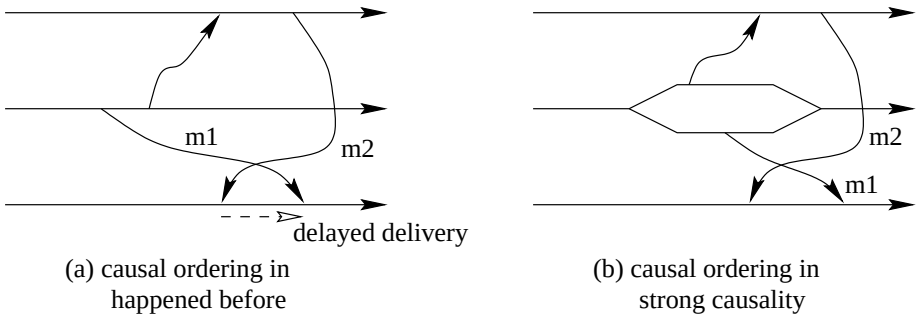


Figure 5: Causal Ordering

5.4 Causal Ordering

While there has been a lot of research in causal ordering, we focus on one paper that criticizes causal ordering [CS93] in the happened before model. As stated in the paper, causal ordering "ensures that messages are delivered in an order that is consistent with the potential causal dependencies between messages".

One of the main criticisms that the authors level against causal ordering in the happened before model is that "false causality reduces performance by unnecessarily delaying messages until the earlier supposedly 'causally related' messages are received and delivered". This situation is shown in Figure 5(a) in which message $m2$ must be delayed until the message $m1$ is received and delivered.

However, it is possible that messages $m1$ and $m2$ were independent (i.e. had independent send events), in which case, this delay was unnecessary. If we remodel the computation in the potential causality model as shown in Figure 5(b), the independence of the two messages becomes apparent and the delay may be avoided.

To summarize: the potential causality model leads to more efficient implementations of causal ordering than the happened before model by avoiding unnecessary delaying of messages owing to false causality.

6 Discussions and Conclusions

A computation, in the sense that we have used it, is something that has actually happened. It may, therefore, seem peculiar that a single potential causality model may capture multiple possible interleavings of local events (or, multiple possible happened before computations). The same fact may be noticed in the happened before model as compared to the real time ordering model. The happened before model allows multiple possible real time orderings. In both cases, the explanation for this seeming paradox is that *what actually happened lies in the eyes of the beholder*. If the observer is interested in applications that require logical time, he would pay attention to the ordering of local events based on real time. However, if the observer were interested in applications that require potential causality, he would only pay attention to the causal ordering of local events.

The crucial deciding factor in choosing the happened before or the potential causality model is the type of application being considered. A litmus test for making this decision would be to ask the following question: *Would a global clock help?* A "yes" would indicate happened before, while a "no" would indicate potential causality. For example, in mutual exclusion, a global clock would allow the timestamping of critical section requests in real time and lead to a fair granting order.

In fact, even without a global clock, if there were multiple threads in a process, we would prefer to treat local events as being totally ordered using the local process clock. In an application like race detection, for example, a global clock would have no advantage because any more ordering would merely induce more false causality and reduce our chances of detecting a race (as shown in Section 5).

Sometimes the happened before model has been interpreted in a manner that allows a separate process to model each local (operating system) process or each local thread. This is a model that tries to approximate true causality but falls short of tracking any independences beyond those that stand out as being parallel. However, such an interpretation should be distinguished from the original intent of happened before, which was to model logical time. Thus, mutual exclusion would not use such an interpretation while race detection would.

We have drawn attention to the false causality problem which limits the effectiveness of the happened before model in modeling potential causality. The potential causality model was introduced to correct this problem. We have demonstrated its applicability in a number of application areas.

References

[AHM93] L. Alvisi, B. Hoppe, and K. Marzullo. Nonblocking and orphan-free message logging protocols. In *Proc. of the 23rd IEEE Symposium on Fault-Tolerant Computing Systems*, pages 145 – 154, Toulouse, France, June 1993.

[CS93] D. R. Cheriton and D. Skeen. Understanding the limitations of causally and totally ordered communication. In *Proc. of the 11th ACM Symposium on Operating System Principles*, pages 44 – 57, Austin, USA, 1993.

[EJW96] E. N. Elnozahy, D. B. Johnson, and Y. M. Wang. A survey of rollback-recovery protocols in message-passing systems. Technical Report CMU-CS-96-181, Dept. of Computer Science, Carnegie Mellon University, Pittsburgh, USA, 1996. (also available at ftp://ftp.cs.cmu.edu/user/mootaz/papers/S.ps).

[EZ92] E. N. Elnozahy and W. Zwaenepoel. Manetho: Transparent rollback-recovery with low overhead, limited rollback and fast output commit. *IEEE Transactions on Computers*, 41(5):526 – 531, May 1992.

[GW94] V. K. Garg and B. Waldecker. Detection of weak unstable predicates in distributed programs. *IEEE Transactions on Parallel and Distributed Systems*, 5(3):299 – 307, March 1994.

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558 – 565, July 1978.

[Mat89] F. Mattern. Virtual time and global states of distributed systems. In *Parallel and Distributed Algorithms: Proc. of the International Workshop on Parallel and Distributed Algorithms*, pages 215 – 226. Elsevier Science Publishers B. V. (North Holland), 1989.

[NBDK96] R.H.B. Netzer, T.W. Brennan, and S. K. Damodaran-Kamal. Debugging race conditions in message-passing programs. In *SIGMETRICS Symposium on Parallel and Distributed Tools*, Philadelphia, USA, May 1996.

[SBN+97] S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. Anderson. Eraser: a dynamic data race detector for multi-threaded programs. In *Proc. of the 16th ACM Symposium on Operating System Principles*, pages 27 – 37, Saint-Malo, France, October 1997.

[TG98] A. Tarafdar and V. K. Garg. Addressing false causality while detecting predicates in distributed programs. In *Proc. of the 18th International Conference on Distributed Computing Systems*, pages 94 – 101, Amsterdam, The Netherlands, May 1998.