Information Processing Letters 111 (2011) 447–452

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Repeated detection of conjunctive predicates in distributed executions

Ajay D. Kshemkalyani

University of Illinois at Chicago, Chicago, IL 60607, United States

ARTICLE INFO

Article history:
Received 13 September 2010
Received in revised form 28 December 2010
Accepted 23 January 2011
Available online 4 February 2011
Communicated by G. Chockler

Keywords:
Distributed computing
Predicate detection
Intervals
Monitoring
Causality
Global state

ABSTRACT

Given a conjunctive predicate ϕ over a distributed execution, this paper gives an algorithm to detect *all* interval sets, each interval set containing one interval per process, in which the local values satisfy the *Definitely*(ϕ) modality. The time complexity of the algorithm is $O(n^3 p)$, where n is the number of processes and p is the bound on the number of times a local predicate becomes true at any process. The paper also proves that unlike the *Possibly*(ϕ) modality which admits $O(p^n)$ solution interval sets, the *Definitely*(ϕ) modality admits $O(np)$ solution interval sets. The paper also gives an on-line test to determine whether all solution interval sets can be detected in polynomial time under arbitrary fine-grained causality-based modality specifications.

1. Introduction

Predicate detection over a distributed execution is important for various purposes such as monitoring, synchronization and coordination, debugging, and industrial process control. Due to asynchrony in message transmissions and in local executions, different executions of the same distributed program go through different sequences of global states. We often need to make assertions about all states in all possible executions of a distributed program. Therefore, two modalities have been defined under which a predicate ϕ can hold for a distributed execution [4].

- *Possibly*(ϕ): There exists a consistent observation of the execution such that ϕ holds in a global state of the observation.
- *Definitely*(ϕ): For every consistent observation of the execution, there exists a global state of it in which ϕ holds.

An online centralized algorithm to detect *Possibly*(ϕ) and *Definitely*(ϕ) for an arbitrary predicate ϕ was given in [4]. The algorithm works by building a lattice of global states. Although it detects generalized global predicates, the time complexity of the algorithm is e^n, where e is the maximum number of events on any process, and n is the number of processes. To reduce the complexity of the algorithm, researchers focused on special classes of global predicates. Conjunctive global predicates form a popular class for many applications [11], and they can be detected under these modalities in polynomial time. This paper considers only conjunctive predicates.

For *conjunctive* predicates, there are time intervals at each process during which the local predicate is true. A global solution under the *Possibly* or *Definitely* modality identifies $\mathcal{I}$, a set of intervals, containing one interval per process in which the local predicate is true, such that the intervals in $\mathcal{I}$ are related by the modality. During such intervals, actual values of the variables, those in consecutive local states, and those in the corresponding composite global states, do not matter [1,5–8,17]. (Identifying each composite global state in a set of intervals is relevant more for non-conjunctive predicates, for which the algorithm in [4] or more efficient techniques like computation slicing [15,16] can be used.)

E-mail address: ajay@uic.edu.

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2011.01.016

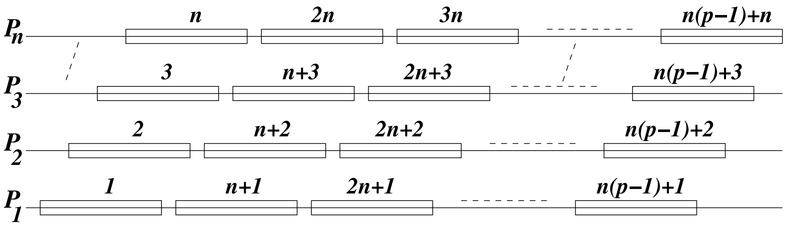

Fig. 1. Example execution using a timing diagram to illustrate the bound on the number of solution interval sets. The message-passing is not shown.

For an execution in which a local predicate becomes true at most p times at a process and n is the number of processes, the best algorithms for detecting *Possibly*(ϕ) [6] and *Definitely*(ϕ) [7] have time complexity $O(n^2p)$ at a central server process. Several distributed algorithms have also been proposed, e.g., [1,5,8,17]. However, all these algorithms detect only the *first interval set* in which ϕ is satisfied under the modality.

We address the problem of identifying *all* solution interval sets $\mathcal{I}$ in a distributed execution that satisfy the *Definitely* modality, not just the first solution set. This problem arises in sensing applications where the monitor program has to raise an alarm each time a predicate becomes true under a certain modality. For example, (i) `reset thermostat to 27deg each time "motion detected" ∧ "temp > 30deg" becomes true`; (ii) `lock the office_door each time "lights off" ∧ "no motion detected" becomes true`; and (iii) `raise alarm each time "stock_S > 85" ∧ "commodity_C ⩽ 20" becomes true`. This problem cannot be solved by simply re-executing the algorithms [6,7] to detect the modality (*Possibly* or *Definitely*, respectively). To appreciate this, consider an example execution, such as that in Fig. 1, in which there is no message communication, or messages might be sent after each interval but asynchronously reach other processes at the end of the execution. In this case, it is necessary for each interval to be considered as a possible candidate for inclusion in a global solution set $\mathcal{I}$. It is not hard to observe that there are p^n "interval sets" in the state–interval lattice. Under the *Possibly* modality, all of these interval sets are solutions to our problem – hence enumerating them will cost $\Omega(p^n)$ time. The current algorithm for detecting the first solution that satisfies *Possibly* (running in $O(n^2p)$) is clearly inadequate.

We note that the algorithm for detecting *Definitely* is very similar to that for *Possibly* and both cost $O(n^2p)$ to detect the *first* solution set. Although we cannot polynomially detect all solution sets for *Possibly*, this paper proposes an algorithm that detects *every* solution set that satisfies *Definitely* in $O(n^3p)$ time. We also prove that there are only $O(np)$ solutions (interval sets) that can satisfy the predicate under the *Definitely* modality, unlike the case for the *Possibly* modality which admits up to $O(p^n)$ solution sets.

2. Model and background

We assume an asynchronous distributed system in which n processes communicate by reliable message passing. Messages may be delivered out of order on the channels. A poset event structure model $(E, \rightarrow)$, where $\rightarrow$ is an irreflexive partial ordering representing the causality relation [12] on the event set E, is used as the model for a distributed system execution. Three kinds of events are considered: send, receive, and internal events. E is partitioned into local executions at each process. Let N denote the set of all processes. Each E_i is a totally ordered set of events executed by process P_i. We assume vector clocks are available [13,14]. Each process maintains a vector clock V of size $n = |N|$ integers, by using the following rules. (1) Before an internal event at process P_i, the process P_i executes $V_i[i] = V_i[i] + 1$. (2) Before a send event at process P_i, the process P_i executes $V_i[i] = V_i[i] + 1$. It then sends the message timestamped by V_i. (3) When process P_j receives a message with timestamp T from process P_i, P_j executes $(\forall k \in [1, \ldots, n])$ $V_j[k] = \max(V_j[k], T[k])$; $V_j[j] = V_j[j]+1$ before delivering the message. The timestamp of an event is the value of the vector clock when the event occurs.

A *conjunctive predicate* $\phi = \bigwedge_i \phi_i$, where ϕ_i is a predicate defined on variables local to process P_i. Let us define durations of interest at each process as the durations in which the local predicate is true. Such an interval at process P_i is identified by the (totally ordered) subset of adjacent events of E_i for which the predicate is true. We use $V_i^-(X)$ and $V_i^+(X)$ to denote the vector timestamp for interval X at process P_i at the start and the end of X, respectively.

We assume that intervals X and Y occur at P_i and P_j, respectively, and are denoted as X_i and Y_j, respectively. We also assume that there are a maximum of p intervals at any process. For any two intervals X and X' that occur at the same process, if X ends before X' begins, we say that X' is a *successor* of X and denote it as $X' = succ(X)$.

For intervals X and Y, we define: $X \hookrightarrow Y$ iff $\exists x \in X, \exists y \in Y, x \rightarrow y$. The relation $\hookrightarrow$ is used by the algorithm to detect *Definitely*(ϕ). In terms of vector timestamps, $X_i \hookrightarrow Y_j$ iff $V_i^-(X_i)[i] \leqslant V_j^+(Y_j)[i]$.

The following two results [7,9] are used in the context of detecting *Definitely*(ϕ).

Theorem 1. *Let $\phi_{i,j} = \phi_i \wedge \phi_j$. Definitely($\phi_{i,j}$) holds if and only if $X_i \hookrightarrow Y_j$ and $Y_j \hookrightarrow X_i$.*

Theorem 1 holds when the local predicate is false in the initial state and final state. To uphold the theorem when ϕ_i is true in these states, one can engineer as follows. When ϕ_i is true in the initial state, P_i broadcasts a control message that is received by all in their initial states, inducing

```
type Log
    start: array[1...n] of integer
    end: array[1...n] of integer

queue of Log: Q_1, Q_2, ... Q_n = ⊥
set of int: updatedQueues, newUpdatedQueues = ∅
int: MaxVector[1...n]
int: count

When an interval begins:
Log_i.start = V_i^-
When an interval ends:
Log_i.end = V_i^+
Send Log_i to P_0

On receiving an interval from process P_z at P_0:
(1) Enqueue the interval onto queue Q_z
(2) if (number of intervals on Q_z is 1) then
(3)     updatedQueues = {z}
(4)     while (updatedQueues ≠ ∅)
(5)         newUpdatedQueues = ∅
(6)         for each i ∈ updatedQueues
(7)             if (Q_i is non-empty) then
(8)                 X = head of Q_i
(9)                 for j = 1 to n (i ≠ j)
(10)                    if (Q_j is non-empty) then
(11)                        Y = head of Q_j
(12)                        if X.end[j] < Y.start[j] then      // test X.end[i] < Y.start[i] for Possibly
(13)                            newUpdatedQueues = {i} ∪ newUpdatedQueues
(14)                        if Y.end[i] < X.start[i] then      // test Y.end[j] < X.start[j] for Possibly
(15)                            newUpdatedQueues = {j} ∪ newUpdatedQueues
(16)        Delete heads of all Q_k, where k ∈ newUpdatedQueues
(17)        updatedQueues = newUpdatedQueues
(18)        if (all queues are non-empty) and (updatedQueues = ∅) then
(19)            Heads of queues identify intervals that form solution set I
(20)            for k = 1 to n
(21)                MaxVector[k] = head(Q_k).end[k]
(22)            for k = 1 to n
(23)                count = 0
(24)                for l = 1 to n (l ≠ k)
(25)                    if head(Q_k).end[l] < MaxVector[l] then
(26)                        count++
(27)                if count = n − 1 then
(28)                    newUpdatedQueues = {k} ∪ newUpdatedQueues
(29)            Delete heads of all Q_k, where k ∈ newUpdatedQueues
(30)            updatedQueues = newUpdatedQueues
```

Fig. 2. On-line algorithm at the data fusion server P_0 to detect all solution interval sets that satisfy *Definitely* for a conjunctive predicate.

the $\rightarrow$ relation. Analogously, when ϕ_i is true in the final state (and no messages were sent since it became true), P_i broadcasts a control message that is received by all in the final state.

Theorem 2. *For a conjunctive predicate ϕ, Definitely(ϕ) holds if and only if Definitely($\phi_{i,j}$) is true for all process pairs P_i and P_j in N.*

Problem statement. In a distributed execution, identify *each* set $\mathcal{I}$ of intervals, containing one interval from each process, such that (i) the local predicate ϕ_i is true in $I_i \in \mathcal{I}$, and (ii) for each pair of processes P_i and P_j, $I_i \hookrightarrow I_j$ and $I_j \hookrightarrow I_i$ are true, i.e., *Definitely*($\phi_{i,j}$) holds.

3. Algorithm

The algorithm is given in Fig. 2. Lines (1)–(19) include the logic to find the *first* solution $\mathcal{I}$ for *Definitely*(ϕ), based on [7]. This code "terminates" when the *first* solution is found and the intervals at the heads of the queues form $\mathcal{I}$. However, intervals in this solution may be part of other solutions that also satisfy *Definitely*(ϕ). The challenge for detecting *all* solutions is two-fold.

1. *Polynomial solvability test*: To determine whether any of these intervals at the heads of the queues can be deleted, or need to be retained because they can all be parts of other solutions (as is the case for *Possibly*). If the head of even one queue cannot be safely deleted,

then the algorithm to detect all interval sets that satisfy the modality may take exponential time.

2. *Identifying intervals for deletion*: If any of these intervals in the solution set, that are now at the heads of their queues, can be deleted, then to identify and delete such intervals.

Given X_i, Y_j in a solution $\mathcal{I}$, we have *Definitely*(X_i, Y_j). An interval $X_i \in \mathcal{I}$ cannot be deleted from $head(Q_i)$ if it is potentially part of another solution, i.e., *Definitely*$(X_i, succ(Y_j))$ may potentially be true for any $Y_j \in \mathcal{I}$. Equation 1 expresses *Definitely*$(X_i, succ(Y_j))$ in terms of timestamps of X_i and $succ(Y_j)$.

$$\begin{aligned}
&\mathit{Definitely}\big(X_i, succ(Y_j)\big)\\
&\quad \Leftrightarrow X_i \hookrightarrow succ(Y_j) \wedge succ(Y_j) \hookrightarrow X_i\\
&\quad \Leftrightarrow true \wedge succ(Y_j) \hookrightarrow X_i\\
&\qquad //X_i \hookrightarrow Y_j \Rightarrow X_i \hookrightarrow succ(Y_j)\\
&\quad \Leftrightarrow V^-\big(succ(Y_j)\big)[j] \leqslant V^+(X_i)[j] \qquad (1)
\end{aligned}$$

Then, if $\forall Y_j (j \neq i) \in \mathcal{I}$, the right-hand side (R.H.S.) of Eq. (1) is false, we have that $\forall j (j \neq i), succ(Y_j) \not\hookrightarrow X_i$. Hence *Definitely*$(X_i, succ(Y_j))$ is false for all $Y_j \in \mathcal{I}$, and X_i can safely be deleted because it cannot overlap with the successor of any other interval in the current solution. So we have:

$$\begin{aligned}
&dequeue\big(head(Q_i)\big) \quad \text{iff}\\
&\forall Y_j (j \neq i) \in \mathcal{I}, V^-\big(succ(Y_j)\big)[j] > V^+(X_i)[j] \qquad (2)
\end{aligned}$$

Eq. (2) expresses the timestamp test for deleting the interval at the head of a queue. A drawback of this test is that the timestamps of the successors of Y_j are needed. As we do not know the values of $V^-(succ(Y_j))[j]$ for all the future intervals $succ(Y_j)$, and we would like to prune all the queues (e.g., Q_i) as soon as possible, we use the following fact that expresses "the start timestamp of any successor of Y_j is greater than the end timestamp of Y_j":

$$V^-\big(succ(Y_j)\big)[j] > V^+(Y_j)[j] \qquad (3)$$

Eq. (3), in conjunction with the timestamp test in the R.H.S. of Eq. (2), gives the implication:

$$\begin{aligned}
&V^+(Y_j)[j] > V^+(X_i)[j]\\
&\quad \Rightarrow V^-\big(succ(Y_j)\big)[j] > V^+(X_i)[j] \qquad (4)
\end{aligned}$$

This implication allows us to use the following approximation, (that uses only timestamps of intervals in $\mathcal{I}$, instead of those of all successor intervals), to determine whether it is safe to dequeue $X_i \in \mathcal{I}$ from Q_i of Eq. (2).

$$\begin{aligned}
&dequeue\big(head(Q_i)\big) \quad \text{iff}\\
&\forall Y_j (j \neq i) \in \mathcal{I}, V^+(Y_j)[j] > V^+(X_i)[j] \qquad (5)
\end{aligned}$$

The approximation of Eq. (5), expressed in terms of timestamps of intervals, is implemented in the algorithm, lines (20)–(30). The code of lines (18)–(30) can also be decentralized and used to repeatedly detect solution interval sets in conjunction with the distributed algorithm in [1].

If the R.H.S. of Eq. (5) is satisfied, then the R.H.S. of Eq. (2) is satisfied, and X_i is dequeued safely. On the other hand, if the R.H.S. of Eq. (5) is not satisfied but the R.H.S. of Eq. (2) is satisfied, then X_i is not dequeued due to the approximation of Eq. (5) that is implemented instead of the accurate condition of Eq. (2).

4. Correctness and complexity

The interval set forming the first solution is correctly detected using the logic of lines (1)–(19).

Theorem 3 *(Safety). Once a solution $\mathcal{I}$ is detected, only intervals $X_i \in \mathcal{I}$ that cannot be part of another solution are deleted from their queues.*

Proof. The algorithm deletes only those intervals in lines (20)–(30) that satisfy the R.H.S. of Eq. (5), and hence the R.H.S. of Eq. (2). These intervals are never going to be part of another solution. Therefore, even if the R.H.S. of Eq. (5) is an approximation to the R.H.S. of Eq. (2), it guarantees safety in dequeuing. □

The next solution is again found by the logic of lines (1)–(19).

The following theorem is useful to show that all solutions can be detected in polynomial time.

Theorem 4 *(Liveness). For any solution set $\mathcal{I}$, at least one interval gets deleted from its queue.*

Proof. We take recourse to a global time axis. Let $X_i \in \mathcal{I}$ be that interval that finishes earliest and let Y_j be any other interval in the solution set $\mathcal{I}$. Such an X_i must satisfy Eq. (5) because $\forall j$, $V_j[j]$ ticks when Y_j completes and hence $Y_j.end$ happens later in global time than $X_i.end$; implying that $Y_j.end[j] \nleq X_i.end[j]$. Hence such an X_i gets deleted in lines (20)–(30).

Therefore, even if the R.H.S. of Eq. (5) is an approximation to the R.H.S. of Eq. (2), it guarantees liveness by way of dequeuing member(s) from $\mathcal{I}$. □

Theorem 5. *The number of solution sets for Definitely(ϕ) for a conjunctive predicate ϕ is bounded by $n(p-1)+1$.*

Proof. From Theorem 4, the number of solution interval sets is bounded by the total number of intervals, viz., np. As a solution set contains n intervals, this bound is more accurately stated as $n(p-1)+1$.

Fig. 1 gives an example execution where this bound is achieved. The rectangles denote the local intervals. Messages are sent and received at least once from each interval to each overlapping interval, but are not depicted in the figure to keep it simple. In this example, the intervals numbered $\{x, x+1, \ldots, x+n-1\}$ form a solution set, for all $x \in [1, n(p-1)+1]$. □

Theorem 6. *All solution sets satisfying Definitely(ϕ) for a conjunctive predicate ϕ can be detected in $O(n^3 p)$ time.*

Proof. Let k be the total number of steps executed, and let $s \in [0, n(p-1)+1]$ be the actual number of solution interval sets. Each interval at the head of a queue incurs a cost c of $O(n)$ due to the role of X in line (8) and the ensuing loop of line (9). Thus, $k/c = np$, the total number of intervals, and $k = O(n^2 p)$ to find zero or one solution in the whole execution. We refine this to account for the cost of detecting all solution sets.

For each solution interval set $\mathcal{I}$,

- Each interval at the head of a queue incurs a cost of $O(n)$ due to the role of X in line (8) and the ensuing loop of line (9). A time cost of n^2 is incurred to find a solution;
- To dequeue at least one interval from $\mathcal{I}$, time cost is n^2 in lines (20)–(30).

Thus the total number of execution steps for processing the intervals in one $\mathcal{I}$ are $O(2n^2)$.

Then for every c $(=n)$ operations out of $k - (2n^2)s$ operations, one interval must get deleted from the head of its queue as it does not go towards forming a solution. We thus have $\frac{k-(2n^2)s}{c} = np$. As $s \leqslant n(p-1)+1$, we have k maximized at $k = O(n^2 p) + O(n^3 p) = O(n^3 p)$.

In essence, at $O(n^2)$ cost, at least one interval gets deleted from some solution set; as there are up to a maximum of *np* solution interval sets, the upper bound on time complexity is $O(n^3 p)$. □

However, as explained by a counter-example in Section 1, for *Possibly*(ϕ), the number of solution sets is $O(p^n)$ even though the algorithm is very similar to that for *Definitely*(ϕ); only the tests in lines (12) and (14) are different as shown in the comments of Fig. 2.

5. Discussion

We now extend our analysis of the condition(s) for repeated detection of conjunctive predicates in polynomial time, for a wide class of modalities, besides *Possibly* and *Definitely*. The approach is a generalization of that in Section 3.

Possibly and *Definitely* are two special cases of fine-grained modalities on predicates, as shown in [10] using the theory in [9]. Any pair of intervals at two processes can be related in only one way out of a *complete* set $\Re$ of 40 possible *orthogonal* ways. These 40 relations come in pairs; if $R(X, Y)$ then $R^{-1}(Y, X)$. For each pair of processes (P_i, P_j), we can specify a set $r^*_{ij} \subseteq \Re$ such that some relation in r^*_{ij} for that P_i and P_j must hold in a solution. Now consider the objective where we need to identify one interval per process such that some relation in r^*_{ij} must hold for each (P_i, P_j) process pair. This gives rise to a problem specification space of size $(2^{40}-1)C^n_2$, of which *Possibly* and *Definitely* are only two special cases. This was formalized as the problem *Fine_Rel′* in [2] and a $O(n^2 p)$ time algorithm was given to detect the solution. The theory was further extended and distributed algorithms were given in [3] to solve this problem. Polynomial time solutions were possible only under a certain condition that was specified using the *prohibition function*.

Definition 1. For each $r_{ij} \in \Re$, prohibition function $\mathcal{H}(r_{ij}) = \{R \in \Re \mid \text{if } R(X_i, Y_j) \text{ is true, then } r_{ij}(X_i, succ(Y_j)) \text{ is false for all } succ(Y_j)\}$.

If for each r^*_{ij}, the following CONVEXITY property held, then a polynomial time solution to problem *Fine_Rel′* was possible.

Definition 2. CONVEXITY: $\forall R \notin r^*_{ij}$: $(\forall r_{ij} \in r^*_{ij}, R \in \mathcal{H}(r_{ij}) \vee \forall r_{ji} \in r^*_{ji}, R^{-1} \in \mathcal{H}(r_{ji}))$.

The CONVEXITY property was necessary and sufficient to detect the *first* solution in polynomial time. We can observe that for the *Fine_Rel′* modalities, the CONVEXITY property will not hold for detecting *all* solutions in polynomial time. Once a solution set $\mathcal{I}$ is detected (using the algorithms in [2,3]), we need to be able to safely prune at least one of the intervals in $\mathcal{I}$ to avoid queue build-up, analogous to the first challenge in Section 3. Define $R^{\mathcal{I}}_{ij}(X_i, Y_j) \in r^*_{ij}$ to be the relation from $\Re$ that holds between intervals $X_i, Y_j \in \mathcal{I}$. Then, we formulate the following analog of Eq. (2), in terms of the above theory and without using timestamps.

$$dequeue\big(head(Q_i)\big) \quad \text{iff}$$

$$\forall Y_j (j \neq i) \in \mathcal{I}, R^{\mathcal{I}}_{ij} \in \bigcap_{r_{ij} \in r^*_{ij}} \mathcal{H}(r_{ij}) \tag{6}$$

The interval X_i at the head of Q_i can be dequeued only if the R.H.S. of Eq. (6) holds. Informally, we can dequeue X_i if, for every other process P_j, X_i will not satisfy any of the relations in r^*_{ij} with any $succ(Y_j)$ interval. Assuming $R^{\mathcal{I}}_{ij}(X_i, Y_j)$ has been determined while detecting the solution, the additional cost of executing the test in Eq. (6) for all $i \in N$ is $O(n^2)$. Note that the test can be executed only at run-time because we do not know beforehand which $R^{\mathcal{I}}_{ij}(X_i, Y_j) \in r^*_{ij}$ will hold in a particular solution $\mathcal{I}$. Simply using the input specification of r^*_{ij} and checking for each $R \in r^*_{ij}$ in Eq. (6), instead of checking for the actual $R^{\mathcal{I}}_{ij}(X_i, Y_j)$ is an over-kill and gives false negatives for the polynomial solvability test.

References

[1] P. Chandra, A.D. Kshemkalyani, Distributed algorithm to detect strong conjunctive predicates, Information Processing Letters 87 (5) (September 2003) 243–249.
[2] P. Chandra, A.D. Kshemkalyani, Causality-based predicate detection across space and time, IEEE Transactions on Computers 54 (11) (2005) 1438–1453.
[3] P. Chandra, A.D. Kshemkalyani, Data stream based global event monitoring using pairwise interactions, Journal of Parallel and Distributed Computing 68 (6) (2008) 729–751.
[4] R. Cooper, K. Marzullo, Consistent detection of global predicates, in: Proc. ACM/ONR Workshop on Parallel and Distributed Debugging, May 1991, pp. 163–173.
[5] V.K. Garg, C. Chase, Distributed detection of conjunctive predicates, in: Proc. IEEE International Conference on Distributed Computing Systems, June 1995, pp. 423–430.
[6] V.K. Garg, B. Waldecker, Detection of weak unstable predicates in distributed programs, IEEE Trans. Parallel and Distributed Systems 5 (3) (Mar. 1994) 299–307.

[7] V.K. Garg, B. Waldecker, Detection of strong unstable predicates in distributed programs, IEEE Trans. Parallel and Distributed Systems 7 (12) (Dec. 1996) 1323–1333.
[8] M. Hurfin, M. Mizuno, M. Raynal, M. Singhal, Efficient distributed detection of conjunctions of local predicates, IEEE Transactions on Software Engineering 24 (8) (1998) 664–677.
[9] A.D. Kshemkalyani, Temporal interactions of intervals in distributed systems, Journal of Computer and System Sciences 52 (2) (April 1996) 287–298.
[10] A.D. Kshemkalyani, A fine-grained modality classification for global predicates, IEEE Transactions on Parallel and Distributed Systems 14 (8) (August 2003) 807–816.
[11] A.D. Kshemkalyani, M. Singhal, Distributed Computing: Principles, Algorithms, and Systems, Cambridge University Press, 2008.
[12] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Communications of the ACM 21 (7) (July 1978) 558–565.
[13] F. Mattern, Virtual time and global states of distributed systems, in: Parallel and Distributed Algorithms, North-Holland, 1989, pp. 215–226.
[14] M. Raynal, M. Singhal, Logical time: capturing causality in distributed systems, IEEE Computer 29 (2) (Feb. 1996) 49–56.
[15] A. Sen, V.K. Garg, On checking whether a predicate definitely holds, in: Proc. 3rd Int. Workshop on Formal Approaches to Testing of Software (FATES 2003), in: Lecture Notes in Computer Science, vol. 2931, Springer, 2004, pp. 15–29.
[16] A. Sen, V.K. Garg, Formal verification of simulation traces using computation slicing, IEEE Transactions on Computers 56 (4) (2007) 511–527.
[17] S. Stoller, F. Schneider, Faster possibility detection by combining two approaches, in: Proc. 9th International Workshop on Distributed Algorithms, in: Lecture Notes in Computer Science, vol. 972, Springer, 1995, pp. 318–332.