Repeated detection of conjunctive predicates in distributed executions

Ajay D. Kshemkalyani

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

Abstract

Given a conjunctive predicate $\phi$ 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($\phi$) modality. The time complexity of the algorithm is $O(n^3p)$, 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($\phi$) modality which admits $O(pn)$ solution interval sets, the Definitely($\phi$) 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.

© 2011 Elsevier B.V. All rights reserved.

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 $\phi$ can hold for a distributed execution [4].

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

An online centralized algorithm to detect Possibly($\phi$) and Definitely($\phi$) for an arbitrary predicate $\phi$ 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 $I$, a set of intervals, containing one interval per process in which the local predicate is true, such that the intervals in $I$ 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.)
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 \( \text{Possibly}(\phi) \) [6] and \( \text{Definitely}(\phi) \) [7] have time complexity \( \Omega(n^2p) \) 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 solution set. Although we cannot polynomially detect all solution sets for \( \text{Possibly} \), this paper proposes an algorithm that detects every solution set that satisfies \( \text{Definitely} \) in \( O(n^2p) \) time. We also prove that there are only \( O(np) \) solutions (interval sets) that can satisfy the predicate under the \( \text{Definitely} \) modality, unlike the case for the \( \text{Possibly} \) modality which admits up to \( \Omega(n^p) \) 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 \( \text{V} \) 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 \( \forall k \in [1,\ldots,n] \), \( V_j[k] = \text{max}(V_j[k], T[k]) \);
   \( V_j[j] = V_j[j] + 1 \) 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 \( \phi_i \) 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 \( X' = \text{succ}(X) \).

For intervals \( X \) and \( Y \), we define: \( X \leftrightarrow Y \) iff \( \exists k \in X, \exists l \in Y, x \rightarrow y \). The relation \( \leftrightarrow \) is used by the algorithm to detect \( \text{Definitely}(\phi) \). In terms of vector timestamps, \( X_i \leftrightarrow Y_j \) iff \( V^-_i(X_i)[i] \leq V^+_j(Y_j)[j] \).

The following two results [7,9] are used in the context of detecting \( \text{Definitely}(\phi) \).

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

Theorem 1 holds when the local predicate is false in the initial state and final state. To uphold the theorem when \( \phi_i \) is true in these states, one can engineer as follows. When \( \phi_i \) is true in the initial state, \( P_i \) broadcasts a control message that is received by all in their initial states, inducing
Theorem 2. For a conjunctive predicate $\phi$, $\text{Definitely}(\phi)$ holds if and only if $\text{Definitely}(\phi_{i,j})$ is true for all process pairs $P_i$ and $P_j$ in $N$.

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

3. Algorithm

The algorithm is given in Fig. 2. Lines (1)–(19) include the logic to find the first solution $I$ for $\text{Definitely}(\phi)$, based on [7]. This code “terminates” when the first solution is found and the intervals at the heads of the queues form $I$. However, intervals in this solution may be parts of other solutions (as is the case for $\text{Possibly}$). 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 $\text{Possibly}$).

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

$$\text{Definitely}(X_i, \text{succ}(Y_j))$$

$$\Leftrightarrow X_i \leftrightarrow \text{succ}(Y_j) \land \text{succ}(Y_j) \leftrightarrow X_i$$

$$\Leftrightarrow \text{true} \land \text{succ}(Y_j) \leftrightarrow X_i$$

$$\text{// } X_i \leftrightarrow Y_j \Rightarrow X_i \leftrightarrow \text{succ}(Y_j)$$

$$\Leftrightarrow V^{-}(\text{succ}(Y_j))[j] \leq V^{+}(X_i)[j]$$  \hfill (1)

Then, if $\forall Y_j(j \neq i) \in I$, the right-hand side (R.H.S.) of Eq. (1) is false, we have that $V(j \neq i), \text{succ}(Y_j) \not\leftrightarrow X_i$. Hence $\text{Definitely}(X_i, \text{succ}(Y_j))$ is false for all $Y_j \in I$, 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:

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

$$\forall Y_j(j \neq i) \in I, V^{-}(\text{succ}(Y_j))[j] > V^{+}(X_i)[j]$$  \hfill (2)

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 $V^{-}(\text{succ}(Y_j))[j]$ for all the future intervals $\text{succ}(Y_j)$, 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 $Y_j$":

$$V^{-}(\text{succ}(Y_j))[j] > V^{+}(Y_j)[j]$$  \hfill (3)

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

$$V^{+}(Y_j)[j] > V^{+}(X_i)[j]$$

$$\Rightarrow V^{-}(\text{succ}(Y_j))[j] > V^{+}(X_i)[j]$$  \hfill (4)

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

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

$$\forall Y_j(j \neq i) \in I, V^{+}(Y_j)[j] > V^{+}(X_i)[j]$$  \hfill (5)

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 $I$ is detected, only intervals $X_i \in I$ 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. $\square$

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 $I$, at least one interval gets deleted from its queue.

**Proof.** We take recourse to a global time axis. Let $X_i \in I$ be that interval that finishes earliest and let $Y_j$ be any other interval in the solution set $I$. Such an $X_i$ must satisfy Eq. (5) because $V(j \neq i), \text{succ}(Y_j) \not\leftrightarrow X_i$, and hence $\text{Definitely}(X_i, \text{succ}(Y_j))$ is false for all $Y_j \in I$, 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:

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

$$\forall Y_j(j \neq i) \in I, V^{-}(\text{succ}(Y_j))[j] > V^{+}(X_i)[j]$$  \hfill (6)

Eq. (6) 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 $V^{-}(\text{succ}(Y_j))[j]$ for all the future intervals $\text{succ}(Y_j)$, 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 $Y_j$”:

$$V^{-}(\text{succ}(Y_j))[j] > V^{+}(Y_j)[j]$$  \hfill (7)

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

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

$$\forall Y_j(j \neq i) \in I, V^{+}(Y_j)[j] > V^{+}(X_i)[j]$$  \hfill (8)

The approximation of Eq. (8), 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].

**Theorem 5.** The number of solution sets for $\text{Definitely}(\phi)$ for a conjunctive predicate $\phi$ 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., $n \cdot p$. 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 $x \in [1, n(p - 1) + 1]$. $\square$

**Theorem 6.** All solution sets satisfying $\text{Definitely}(\phi)$ for a conjunctive predicate $\phi$ can be detected in $O(n^3p)$ time.
Thus have its queue as it does not go towards forming a solution. We iterations, one interval must get deleted from the head of the theory in [9]. Any pair of intervals at two processes tion 3.

Possibly

\[ \forall Y_j (j \neq i) \in I, R_{ij}^T \subseteq \bigcap_{r_{ij} \in T} H(r_{ij}) \quad (6) \]

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

\[ \text{dequeue}(\text{head}(Q_i)) \text{iff} \]

\[ \forall Y_j (j \neq i) \in I, R_{ij}^T \subseteq \bigcap_{r_{ij} \in T} H(r_{ij}) \]

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 \( \text{succ}(Y_j) \) interval. Assuming \( R_{ij}^Z(X_i, Y_j) \) 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 \( R_{ij}^Z(X_i, Y_j) \) will hold in a particular solution \( I \). 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 \( R_{ij}^Z(X_i, Y_j) \) is an over-kill and gives false negatives for the polynomial solvability test.

References


