Detecting Global Predicates in Distributed Systems with Clocks

Scott D. Stoller[⋆]

Dept. of Computer Science, Indiana University, Bloomington, IN 47405, USA

Abstract. This paper proposes a framework for predicate detection in systems of processes with approximately-synchronized real-time clocks. Timestamps from these clocks are used to define two orderings on events: "definitely occurred before" and "possibly occurred before". These orderings lead naturally to definitions of 3 distinct *detection modalities*, *i.e.*, 3 meanings of "predicate Φ held during a computation", namely: $\mathbf{Poss_T}\,\Phi$ ("Φ possibly held"), $\mathbf{Def_T}\,\Phi$ ("Φ definitely held"), and $\mathbf{Inst}\,\Phi$ ("Φ definitely held at a specific instant"). This paper defines these modalities and gives efficient algorithms for detecting them; the algorithms are based on algorithms of Cooper and Marzullo, Garg and Waldecker, and Fromentin and Raynal.

Keywords: global predicate detection, consistent global states, partially-synchronous systems, distributed debugging, real-time monitoring

1 Introduction

A *history* of a distributed system can be modeled as a sequence of events in their order of occurrence. Since execution of a particular sequence of events leaves the system in a well-defined global state, a history uniquely determines a sequence of global states through which the system has passed. Unfortunately, in a distributed system without perfect synchronization, it is, in general, impossible for a process to determine the order in which events on different processors actually occurred. Therefore, no process can determine unambiguously the sequence of global states through which the system passed. This leads to an obvious difficulty for detecting whether a global state predicate (hereafter simply called a "predicate").

Cooper and Marzullo proposed a solution for asynchronous distributed systems [CM91]. Their solution involves two modalities, which we denote by **Poss** (read "possibly") and **Def** (read "definitely"). These modalities are based on logical time [Lam78] as embodied in the *happened-before* relation $\overset{e}{\rightarrow}_{hb}$, a partial ordering[2] of events that reflects causal dependencies. Happened-before is not a total order, so it does not uniquely determine the history, but it does restrict the possibilities. Given a predicate Φ, a computation satisfies $\mathbf{Poss}\,\Phi$ iff there is *some* interleaving of events that is consistent with happened-before and in which the system passes through a global state satisfying Φ. A computation satisfies $\mathbf{Def}\,\Phi$ iff for *every* interleaving of events that is consistent with happened-before, the system passes through a global state satisfying Φ.

[⋆] Email: *stoller@cs.indiana.edu*. Web: *http://www.cs.indiana.edu/~stoller/*

[2] In this paper, all partial orderings are irreflexive unless specified otherwise.

Cooper and Marzullo's definitions of these modalities established an important conceptual framework for predicate detection in asynchronous systems, which has been the basis for considerable research [DJR93, GW94, CBDGF95, JMN95, SS95, GW96]. In practice, though, detection of **Poss** or **Def** suffers from two significant burdens. First, most of the detection algorithms require that each process maintain a vector clock; this imposes computational overhead of $O(N)$ arithmetic operations per "tick" (of the vector clock) and requires that a vector timestamp with $O(N)$ components be attached to each message, where N is the number of processes in the system. Second, detecting **Poss** $\varPhi$ or **Def** $\varPhi$ can be computationally expensive: the worst-case time complexity is $\varOmega(E^N)$, where E is the maximum number of events executed by each process.

This paper proposes a framework for predicate detection in systems with approximately-synchronized real-time clocks. Timestamps from these clocks can be used to define two orderings on events: $\overset{e}{\twoheadrightarrow}$ (read "definitely occurred before") and $\overset{e}{\rightarrow}$ (read "possibly occurred before"). By (roughly speaking) substituting each of these orderings for happened-before in the definitions of **Poss** and **Def**, we obtain definitions of four new modalities. The two modalities based on $\overset{e}{\twoheadrightarrow}$ are closely analogous to **Poss** and **Def**, so we denote them by $\mathbf{Poss_T}$ and $\mathbf{Def_T}$ (the "T" stands for "timed"). We obtain algorithms for detecting $\mathbf{Poss_T}$ and $\mathbf{Def_T}$ by adapting (and, as we do so, optimizing) algorithms of Cooper and Marzullo [CM91] and Garg and Waldecker [GW94, GW96]. Modalities based on $\overset{e}{\rightarrow}$ are quite different, because $\overset{e}{\rightarrow}$ (unlike $\overset{e}{\rightarrow}_{hb}$ and $\overset{e}{\twoheadrightarrow}$) is not a partial ordering. In fact, $\overset{e}{\rightarrow}$ yields a degenerate case, in which the analogues of **Poss** and **Def** are equivalent. We show that this single modality, which we denote by **Inst**, is closely related to Fromentin and Raynal's concept of **Properly** [FR94, FR95], and we adapt for detecting **Inst** an algorithm of theirs for detecting **Properly**.

Our detection framework is applicable to a wide range of systems, since it does not require that clocks be synchronized to within a fixed bound. We assume each event is time-stamped with a time interval, with the interpretation: when that event occurred, the value of every (relevant) clock in the system was in that interval. Implementing such timestamps is straightforward assuming the underlying clock synchronization mechanism provides bounds on the offsets between clocks (the *offset* between two clocks (at some instant) is the difference in their values). For example, such information can be obtained from NTP [Mil95] or the Distributed Time Service in OSF DCE [Tan95].

The quality of clock synchronization affects $\overset{e}{\twoheadrightarrow}$ and $\overset{e}{\rightarrow}$ and therefore affects the results of detection. For example, consider **Inst** $\varPhi$. Informally, a computation satisfies **Inst** $\varPhi$ iff the timestamps imply that there was an instant during the computation when predicate $\varPhi$ held, *i.e.*, iff there is some collection of local states that form a global state satisfying $\varPhi$ and that, based on the timestamps, definitely overlapped in time. Suppose $\varPhi$ actually holds in a global state g that persists for time δ. Whether **Inst** $\varPhi$ holds depends on the quality of synchronization. Roughly, if the clock offsets are known to be smaller than δ, then **Inst** $\varPhi$ holds; otherwise, there is in some cases no way to determine whether the local states in g actually overlapped in time, so **Inst** $\varPhi$ might not hold.

The quality of clock synchronization affects also the cost of detection. For example, consider $\mathbf{Poss_T}\,\Phi$. Informally, a computation satisfies $\mathbf{Poss_T}\,\Phi$ iff there is some collection of local states that form a global state satisfying Φ and that, based on the timestamps, possibly overlapped in time. The larger the bounds on the offsets between clocks, the more combinations of local states possibly overlap. In general, Φ must be evaluated in each such combination of local states. Thus, the larger the bounds on the offsets, the more expensive the detection. If the bounds on the offsets are comparable to or smaller than the mean interval between events that potentially truthify or falsify Φ, then the number of global states that must be checked is comparable to the number of global states that the system actually passed through during execution, which is $O(NE)$. In contrast, the number of global states considered in the asynchronous case is $O(E^N)$.

We expect the above condition on the bounds on the offsets to hold in many systems. In most local-area distributed systems, protocols like NTP can efficiently maintain synchronization of clocks to within a few milliseconds [Mil95]. Even in extremely wide-area distributed systems like the Internet, clock synchronization can usually be maintained to within a few tens of milliseconds [Mil91]. The detection framework and algorithms proposed here are designed to provide a basis for monitoring and debugging applications in such systems.

2 Background

A *local computation*—that is, a computation of a single process—is represented as a sequence of local states and events; thus, a local computation has the form

$$e_1,\ s_1,\ e_2,\ s_2,\ e_3,\ s_3,\ \ldots \tag{1}$$

where the e_α are events, and the s_α are local states. For a local state s, $\mathcal{S}(s)$ and $\mathcal{T}(s)$ denote the start event and terminal event, respectively, of s. For example, in (1), $\mathcal{S}(s_2)$ is e_2, and $\mathcal{T}(s_2)$ is e_3.

A *computation* of a distributed system is a collection of local computations, one per process; we represent such a collection as a function from process names to local computations. We use integers $1, 2, \ldots, N$ as process names. Variables i and j always range over process names. We use $Ev(c)$ and $St(c)$ to denote the sets of all events and all local states, respectively, in a computation c. For convenience, we assume all events and local states in a computation are distinct. For a local state s, $pr(s)$ denotes the process that passes through s. For an event e, $pr(e)$ denotes the process on which e occurs. A *global state* of a distributed system is a collection of local states, one per process, represented as a function from process names to local states. The set of global states of a computation c is denoted $GS(c)$; thus, g is in $GS(c)$ iff for each process i, $g(i)$ is a local state in $c(i)$. We define a reflexive partial ordering $\preceq_G$ on global states by:

$$g \preceq_G g' \ \stackrel{\Delta}{=}\ (\forall i : g(i) = g'(i)\ \vee\ (g(i) \text{ occurs before } g(i'))). \tag{2}$$

Each event e has a timestamp $C(e)$, which is an interval with lower endpoint $C_1(e)$ and upper endpoint $C_2(e)$, with the interpretation: when e occurred, every clock in the system had a value between $C_1(e)$ and $C_2(e)$. We require that the clock synchronization algorithm never decrease the value of a clock. This ensures:

SC1 For every event e, $C_1(e) \le C_2(e)$.

SC2 For every event e with an immediately succeeding event e' on the same process, $C_1(e) \le C_1(e')$ and $C_2(e) \le C_2(e')$.

3 Generic Theory of Consistent Global States

Predicate detection in asynchronous systems is based on the notion of consistent global states (CGSs) [BM93]. Informally, a global state is consistent if it could have occurred during the computation. Recall that an *ideal* of a partial order $\langle S, \prec \rangle$ is a set $I \subseteq S$ such that $(\forall x \in I : \forall y \in S : y \prec x \Rightarrow y \in I)$. Ideals of $\langle Ev(c), \overset{e}{\rightarrow}_{hb} \rangle$ are usually called *consistent cuts*. Recall that for any partial order, the set of its ideals ordered by inclusion ($\subseteq$) forms a lattice [DJR93]. Furthermore, the lattice of CGSs ordered by $\preceq_G$ is isomorphic to the lattice of consistent cuts [SM94, BM93]. This isomorphism has an important consequence for detection algorithms; specifically, it implies that a minimal increase with respect to $\preceq_G$ corresponds to advancing one process by one event, and hence that the lattice of CGSs can be explored by repeatedly advancing one process by one event. This principle underlies detection algorithms of Cooper and Marzullo [CM91] and Garg and Waldecker [GW94, GW96].

In this section, we show that the above theory is not specific to the happened-before relation but rather applies to any partial ordering $\overset{e}{\hookrightarrow}$ on events, provided $\overset{e}{\hookrightarrow}$ is *process-wise-total*, *i.e.*, for any two events e and e' on the same process, if e occurred before e', then $e \overset{e}{\hookrightarrow} e'$. This generalization underlies the detection algorithms in Sections 4 and 5.

Definition of CGSs. Let c be a computation, and let $\overset{e}{\hookrightarrow}$ be a relation on $Ev(c)$. We define a relation $\overset{s}{\hookrightarrow}$ on $St(c)$, with the informal interpretation: $s \overset{s}{\hookrightarrow} s'$ if s ends before s' starts. Formally,

$$s \overset{s}{\hookrightarrow} s' \;\triangleq\; \begin{cases} \mathcal{S}(s) \overset{e}{\hookrightarrow} \mathcal{S}(s') & \text{if } pr(s) = pr(s') \\ \mathcal{T}(s) \overset{e}{\hookrightarrow} \mathcal{S}(s') & \text{if } pr(s) \neq pr(s'). \end{cases} \tag{3}$$

Two local states are *concurrent* if they are not related by $\overset{s}{\hookrightarrow}$. A global state is *consistent* if its constituent local states are pairwise concurrent. Thus, the set of CGSs of computation c with respect to $\overset{e}{\hookrightarrow}$ is

$$CGS^{\overset{e}{\hookrightarrow}}(c) = \{g \in GS(c) \mid \forall i, j : i \neq j \Rightarrow \neg(g(i) \overset{s}{\hookrightarrow} g(j))\}. \tag{4}$$

Note that $CGS^{\overset{e}{\rightarrow}_{hb}}$ is the usual notion of CGSs.

Generic Definitions of **Poss** *and* **Def**. The detection modalities **Poss** and **Def** for asynchronous systems are defined in terms of the lattice of CGSs induced by happened-before. We generalize them as follows.

Poss: A computation c satisfies $\mathbf{Poss}^{\overset{e}{\hookrightarrow}} \Phi$ iff $CGS^{\overset{e}{\hookrightarrow}}(c)$ contains a global state satisfying Φ.

$\mathbf{Def}^{\overset{e}{\hookrightarrow}}$ is defined in terms of paths. A *path* of a partial order $\langle S, \preceq\rangle$ is a sequence[3] σ of distinct elements of S such that $\sigma[1]$ and $\sigma[|\sigma|]$ are minimal and maximal, respectively, with respect to $\preceq$ and such that for all $\alpha < |\sigma|$, $\sigma[\alpha+1]$ is an immediate successor[4] of $\sigma[\alpha]$. Informally, each path in $\langle CGS^{\overset{e}{\hookrightarrow}}(c), \preceq_G\rangle$ corresponds to an order in which the events in the computation could have occurred.

Def: A computation c satisfies $\mathbf{Def}^{\overset{e}{\hookrightarrow}} \Phi$ iff every path of $\langle CGS^{\overset{e}{\hookrightarrow}}(c), \preceq_G\rangle$ contains a global state satisfying Φ.

CGSs and Ideals. When $\overset{e}{\hookrightarrow}$ is a process-wise-total partial ordering of events, there is a natural correspondence between $CGS^{\overset{e}{\hookrightarrow}}$ and ideals of $\langle Ev(c), \overset{e}{\hookrightarrow}\rangle$. One can think of an ideal I as the set of events that have occurred. Executing set I of events leaves each process i in the local state immediately following the last event of process i in I. Thus, ideal I corresponds to the global state g such that for all i, $\mathcal{S}(g(i))$ is the maximal element of $\{e \in I \mid pr(e) = i\}$. This correspondence is an isomorphism.

Theorem 1. *For any process-wise-total partial ordering $\overset{e}{\hookrightarrow}$ on $Ev(c)$, the partial order $\langle CGS^{\overset{e}{\hookrightarrow}}(c), \preceq_G\rangle$ is a lattice and is isomorphic to the lattice of ideals of $\langle Ev(c), \overset{e}{\hookrightarrow}\rangle$.*

Proof. This is true for the same reasons as in the standard theory based on happened-before [SM94, BM93, DJR93]. The proof is straightforward. □

The following corollary underlies the detection algorithms in Sections 4 and 5.

Corollary 2. *If global state g' is an immediate successor of g in $\langle CGS(c), \preceq_G\rangle$, then the ideal corresponding to g' contains exactly one more event than the ideal corresponding to g.*

Proof. This follows from Theorem 1 and the fact that if one ideal of a partial order is an immediate successor of another ideal of that partial order, then those two ideals differ by exactly one element. □

4 Detection Based on a Strong Event Ordering: $\mathbf{Poss_T}$ and $\mathbf{Def_T}$

We instantiate the generic theory in Section 3 with a specific partial ordering $\overset{e}{\twoheadrightarrow}$ ("definitely occurred before"), defined by:

$$e \overset{e}{\twoheadrightarrow} e' \triangleq \begin{cases} \text{e occurs before e'} & \text{if } pr(e) = pr(e') \\ C_2(e) < C_1(e') & \text{if } pr(e) \neq pr(e'). \end{cases} \quad (5)$$

[3] We use 1-based indexing for sequences.

[4] For a reflexive or irreflexive partial order $\langle S, \prec\rangle$ and elements $x \in S$ and $y \in S$, y is an *immediate successor* of x iff $x \neq y \wedge x \prec y \wedge \neg(\exists\, z \in S \setminus \{x, y\} : x \prec z \wedge z \prec y)$.

This ordering cannot be defined solely in terms of the real-time timestamps, since SC1 and SC2 allow consecutive events on a process to have identical timestamps. We solve this problem by assuming that when a process records the real-time timestamp for an event, it records a sequence number $L(e)$ (starting with zero) as well; thus, for events e and e' with $pr(e) = pr(e')$, $e \overset{e}{\twoheadrightarrow} e'$ iff $L(e) < L(e')$.

Theorem 3. *For any computation c, $\overset{e}{\twoheadrightarrow}$ is a process-wise-total partial ordering on $Ev(c)$.*

Proof. See Appendix. □

By the discussion in Section 3, $\overset{e}{\twoheadrightarrow}$ induces an ordering $\overset{s}{\twoheadrightarrow}$ on local states, a notion $CGS^{\overset{e}{\twoheadrightarrow}}$ of CGSs, and detection modalities $\mathbf{Poss}^{\overset{e}{\twoheadrightarrow}}$ and $\mathbf{Def}^{\overset{e}{\twoheadrightarrow}}$, which we denote by $\mathbf{Poss_T}$ and $\mathbf{Def_T}$, respectively. If $g \in CGS^{\overset{e}{\twoheadrightarrow}}(c)$, then the local states in g possibly overlapped in time.

We consider in this paper only detection algorithms with a passive monitor. In such algorithms, each process in the original system sends its timestamped local states to a new process, called the *monitor*. More specifically, for each process, when an event terminating the current local state s occurs, the process sends to the monitor a message containing s and the timestamps $C(\mathcal{S}(s))$ and $C(\mathcal{T}(s))$.[5]

We consider only *on-line* detection, in which the monitor detects the property as soon as possible. Algorithms for *off-line* detection, in which the monitor waits until the computation has terminated and all local states have arrived before checking whether the property is satisfied, can be obtained as special cases. We consider first general algorithms for $\mathbf{Poss_T}$ and $\mathbf{Def_T}$ and then more efficient algorithms that work only for predicates of a certain form.

4.1 General Algorithms for $\mathbf{Poss_T}$ and $\mathbf{Def_T}$

The algorithms in [CM91, MN91] can be adapted to explore lattice $\langle CGS^{\overset{e}{\twoheadrightarrow}}(c), \preceq_G \rangle$ by (roughly) replacing each condition of the form $e \overset{e}{\rightarrow}_{hb} e'$ with $e \overset{e}{\twoheadrightarrow} e'$. Following [CM91, MN91], we give algorithms in which the monitor constructs one level of the lattice of CGSs at a time. The *level* of a global state g is $\sum_{i=1}^{N} L(\mathcal{S}(g(i)))$. Level ℓ of the lattice of CGSs contains the CGSs with level ℓ. Constructing one level of the lattice at a time is unnecessary and sometimes delays detection of a property; this construction is used only to simplify the presentation.

The algorithm used by the monitor to detect $\mathbf{Poss_T}\,\Phi$ is given in Figure 1. To enumerate the states in the next level of the lattice (line 7 of the algorithm),

[5] Several straightforward optimizations are possible. For example, each message might describe only the differences between consecutive reported local states, rather than repeating the entire local state. Also, except for the initial local state, it suffices to include with local state s only the timestamp $C(\mathcal{T}(s))$, since $C(\mathcal{S}(s))$ was sent in the previous message to the monitor. Also, for a given predicate Φ, events that cannot possibly truthify or falsify Φ can be ignored.

the monitor considers each state g in *last* and each process i, and checks whether the next local state s of process i (*i.e.*, the immediate successor on process i of $g(i)$) is concurrent with the local states in g of all the other processes. (The monitor cannot complete construction of the next level until the next local state of each process has arrived.) If so, the monitor adds $g[i \mapsto s]$ to the set *current*, where for a function f and an element x in the domain of f, $f[x \mapsto c]$ is the function that maps x to c and that agrees with f on all arguments except x. Cooper and Marzullo's algorithm for detecting **Def** can be adapted in a similar way to detect $\mathbf{Def_T}$.

```
lvl := 0;
wait until at least one local state has been received from each process;
current := the global state of level 0;
while no state in current satisfies $\Phi$
    last := current;
    lvl := lvl + 1;
    current := consistent global states of level lvl reachable from a state in last
endwhile;
report $\mathbf{Poss_T}\,\Phi$
```

Fig. 1. Algorithm for detecting $\mathbf{Poss_T}\,\Phi$.

Recall that a process sends a local state to the monitor when that local state ends. This is natural (because $\overset{s}{\twoheadrightarrow}$ depends on when local states end) but can delay detection. One approach to bounding and reducing this delay is for a process that has not reported an event to the monitor recently to send a message to the monitor to report that it is still in the same local state (as if the process were reporting that it just executed a "skip" event). Another approach (described in [MN91]) requires knowledge of a bound on message latency: the monitor can use its own local clock and this bound to determine a lower bound on the ending time of the last local state it received from a process.

The time complexity of these algorithms depends on the rate at which events occur relative to the bounds on clock offsets. To simplify the complexity analysis, suppose the offset between two clocks is known to be always at most $\epsilon/2$, so for every event e, $C_2(e) - C_1(e) < \epsilon$. Suppose also that the interval between consecutive events at a process is always at least τ. For each CGS g, the algorithm takes constant time to evaluate Φ and $O(N)$ time to find all of the immediate successors of g in the lattice. If $\tau > \epsilon$, then there are $O(3^N E)$ CGSs, so the worst-case time complexity is $O(3^N NE)$, where E is the maximum number of events executed by any process. If $\tau \leq \epsilon$, then each local state appears in at most $O((\lceil \frac{2\epsilon+\tau}{\tau} \rceil + 1)^{N-1})$ CGSs, so the worst-case time complexity is $O((\lceil \frac{2\epsilon}{\tau} \rceil + 2)^{N-1} NE)$. In both cases, the worst-case time complexity is linear in E, which is normally much larger than N; in contrast, the worst-case time complexity of general algorithms for detecting **Poss** and **Def** is $\Omega(E^N)$.

4.2 Algorithms for $\mathbf{Poss_T}$ and $\mathbf{Def_T}$ for Restricted Predicates

Garg and Waldecker [GW94, GW96] have developed efficient algorithms for detecting $\mathbf{Poss}\,\Phi$ and $\mathbf{Def}\,\Phi$ for conjunctive predicates Φ. A predicate is *conjunctive* if it is a conjunction of predicates that each depend on the local state of one process. Their algorithms can be adapted in a straightforward way to detect $\mathbf{Poss_T}$ and $\mathbf{Def_T}$, by (roughly) replacing comparisons based on happened-before with comparisons based on $\overset{e}{\twoheadrightarrow}$. This yields detection algorithms with worst-case time complexity $O(N^2E)$. The worst-case time complexity of both algorithms can be reduced to $O((N \log N)E)$ by exploiting the total ordering on numbers. We briefly review Garg and Waldecker's algorithm for detecting $\mathbf{Poss}\,\Phi$ for conjunctive predicates and then describe the optimized algorithm for detecting $\mathbf{Poss_T}$ for such predicates.

Suppose the global predicate of interest is $\Phi = \bigwedge_{i=1}^{N} \phi_i$, where ϕ_i depends on the local state of process i. In Garg and Waldecker's algorithm, each process i sends to the monitor timestamped local states satisfying ϕ_i; local states not satisfying ϕ_i are not reported. For each process i, the monitor maintains a queue q_i and adds each timestamped local state received from process i to the end of q_i. Let head(q) denote the head of a non-empty queue q. If for some i and j, head$(q_i) \overset{e}{\rightarrow}_{hb}$ head(q_j), then head(q_i) is removed from q_i. The heads of the queues are repeatedly compared in this way and (when appropriate) removed, until the heads of the non-empty queues are (pairwise) concurrent. Then, if all the queues are non-empty, then the heads of the queues form a CGS satisfying Φ, so the property has been detected; if some queue is empty, then the monitor waits to receive more local states. The worst-case time complexity is $O(N^2E)$, because there are $O(NE)$ local states, and each time a local state is removed from q_i, the new head of q_i is compared with the heads of the other $O(N)$ queues.

For detection of $\mathbf{Poss_T} \bigwedge_{i=1}^{N} \phi_i$, the number of comparisons can be reduced as follows. Expanding the definition of $CGS^{\overset{e}{\twoheadrightarrow}}(c)$, $g \in GS(c)$ is consistent iff

$$(\forall i, j : i \neq j \;\Rightarrow\; C_2(\mathcal{T}(g(i))) \geq C_1(\mathcal{S}(g(j)))). \tag{6}$$

Using the fact that for all i, $C_2(\mathcal{T}(\text{head}(g(i)))) \geq C_1(\mathcal{S}(\text{head}(g(i))))$, which follows from SC1 and SC2, one can show that (6) is equivalent to

$$\min_i(C_2(\mathcal{T}(\text{head}(g(i))))) \;\geq\; \max_i(C_1(\mathcal{S}(\text{head}(g(i))))). \tag{7}$$

To evaluate (7) efficiently, we maintain two priority queues p_1 and p_2, whose contents are determined by the invariants:

I1: For each process i such that q_i is non-empty, p_1 contains a record with key $C_1(\mathcal{S}(\text{head}(q_i)))$ and satellite data i. p_1 contains no other records.

I2: For each process i such that q_i is non-empty, p_2 contains a record with key $C_2(\mathcal{T}(\text{head}(q_i)))$ and satellite data $\langle i, ptr\rangle$, where ptr is a pointer to the record with satellite data i in p_1. p_2 contains no other records.

Recall that the operations on a priority queue p include getMin(p), which returns a record $\langle k, d\rangle$ with key k and satellite data d such that k is the minimal value of the key, and extractMin(p), which removes and returns such a record [CLR90]. We also use priority queues with operations based on maximal key values. Thus, (7) is equivalent to

$$\text{key}(\text{getMin}(p_2)) \geq \text{key}(\text{getMax}(p_1)), \tag{8}$$

where key($\langle k, d\rangle$) $= k$. The negation of (8) is used in the **while** loop in Figure 2 to check whether a CGS has been found. Recall that an operation on a priority queue containing n records takes $O(\log n)$ time. A constant number of such operations are performed for each local state, so the worst-case time complexity of the algorithm in Figure 2 is $O((N \log N)E)$. Note that the time complexity is independent of the rate of events and the quality of clock synchronization.

The algorithm in [GW96] for detecting $\mathbf{Def}\,\Phi$ for conjunctive Φ can be adapted in a similar way to detect $\mathbf{Def_T}\,\Phi$ for such predicates.

```
On receiving $x$ from process $i$:
    append($q_i, x$);
    if head($q_i$) $= x$ then
        add records for $i$ to $p_1$ and $p_2$;
        while $\neg$empty($p_1$) $\wedge$ key(getMin($p_2$)) $<$ key(getMax($p_1$))
            $\langle k, \langle i, ptr\rangle\rangle$ := extractMin($p_2$);
            remove record for $i$ from $p_1$;
            removeHead($q_i$);
            if $\neg$empty($q_i$) then
                add records for $i$ to $p_1$ and $p_2$;
            endif
        endwhile
        if ($\forall i : \neg$empty($q_i$)) then
            report $\mathbf{Poss_T}\,\Phi$ and exit
        endif
    endif
```

Fig. 2. Algorithm for detecting $\mathbf{Poss_T}\,\Phi$ for Φ a conjunction of local predicates.

5 Detection Based on a Weak Event Ordering: Inst

This section considers the ordering "possibly occurred before", defined by:

$$e \overset{e}{\to} e' \;\triangleq\; \begin{cases} \text{e occurs before e'} & \text{if } pr(e) = pr(e') \\ C_1(e) \leq C_2(e') & \text{if } pr(e) \neq pr(e'). \end{cases} \tag{9}$$

Using (3), this induces a relation $\overset{s}{\to}$ on local states, with the interpretation: $s \overset{s}{\to} s'$ if s possibly ended before s' started. Two local states are *strongly concurrent* if they are not related by $\overset{s}{\to}$; such local states must overlap in time. The set $CGS^{\overset{e}{\to}}$ is defined by (4). We call elements of $CGS^{\overset{e}{\to}}$ *strongly consistent* global states (SCGSs).

$\mathbf{Poss}^{\overset{e}{\to}}$ and $\mathbf{Def}^{\overset{e}{\to}}$ are equivalent, *i.e.*, for all computations c and predicates Φ, c satisfies $\mathbf{Poss}^{\overset{e}{\to}}\,\Phi$ iff c satisfies $\mathbf{Def}^{\overset{e}{\to}}\,\Phi$. This equivalence is an easy corollary of the following theorem.

Theorem 4. *$\langle CGS^{\overset{e}{\to}}(c), \preceq_G\rangle$ is a total order (and therefore a lattice).*

Proof. See Appendix. □

We define **Inst** (read "instantaneously") to denote this modality (*i.e.*, $\mathbf{Poss}^{\overset{e}{\to}}$ and $\mathbf{Def}^{\overset{e}{\to}}$). Informally, a computation satisfies **Inst** Φ if there is a global state g satisfying Φ and such that the system definitely passes through g during the computation. Theorem 1 does not apply to $\overset{e}{\to}$, because:

Theorem 5. *$\overset{e}{\to}$ is not a partial ordering.*

Proof. See Appendix. □

Since Theorem 1 does not apply, it is not surprising that a minimal increase in $\langle CGS^{\overset{e}{\to}}(c), \preceq_G\rangle$ does not necessarily correspond to advancing one process by one event (it is easy to construct examples of this). Consequently, the algorithms in Section 4 cannot be easily adapted to detect **Inst**. Our algorithm for detecting **Inst** is based on Fromentin and Raynal's algorithm for detecting **Properly** (read "properly") in asynchronous systems [FR94, FR95]. The definition of **Properly**, generalized to an arbitrary ordering on events, is:

Properly: A computation c satisfies $\mathbf{Properly}^{\overset{e}{\hookrightarrow}}\,\Phi$ iff there is a global state satisfying Φ and contained in every path of $\langle CGS^{\overset{e}{\hookrightarrow}}(c), \preceq_G\rangle$.

Theorem 6. $\mathbf{Properly}^{\overset{e}{\to}}$ *is equivalent to* **Inst**.
Proof. See Appendix. □

As Theorem 6 suggests, Fromentin and Raynal's algorithm for detecting $\mathbf{Properly}^{\overset{e}{\to}_{hb}}$ can be adapted in a straightforward way to detect **Inst**. This yields an algorithm with worst-case time complexity $O(N^3E)$. Optimizations similar to those presented in Section 4.2 are possible here as well. Expanding the definition of $CGS^{\overset{e}{\to}}(c)$, a global state g is strongly consistent iff

$$(\forall i, j : i \neq j \;\Rightarrow\; C_1(\mathcal{T}(g(i))) > C_2(\mathcal{S}(g(j)))). \tag{10}$$

To check this condition efficiently, we introduce priority queues p_1 and p_2, whose contents are determined by the following invariants:

J1: for each process i such that q_i is non-empty, p_1 contains a record with key $C_1(\mathcal{T}(\text{head}(q_i)))$ and satellite data $\langle i, \mathit{ptr}\rangle$, where *ptr* is a pointer to the record with satellite data i in p_2. p_1 contains no other records.
J2: for each process i such that q_i is non-empty, p_2 contains a record with key $C_2(\mathcal{S}(\text{head}(q_i)))$ and satellite data i. p_2 contains no other records.

The goal is to define a condition $\text{SC}(p_1, p_2)$ that tests whether the heads of the non-empty queues are (pairwise) strongly concurrent. Based on (10), a first attempt is $\text{empty}(p_1) \vee \text{key}(\text{getMin}(p_1)) > \text{key}(\text{getMax}(p_2))$. However, this condition is not correct when for some i, $C_1(\mathcal{T}(g(i))) < C_2(\mathcal{S}(g(i)))$. Taking this possibility into account, we obtain

$$\begin{aligned}\mathrm{SC}(p_1,p_2) \triangleq\ & \mathrm{empty}(p_1) \vee \mathrm{key}(\mathrm{getMin}(p_1)) > \mathrm{key}(\mathrm{getMax}(p_2)) \\ & \vee (\pi_1(\mathrm{data}(\mathrm{getMin}(p_1))) = \mathrm{data}(\mathrm{getMax}(p_2)) \wedge \mathrm{countMax}(p_2) = 1),\end{aligned} \tag{11}$$

where countMax(p) is the number of records containing the maximal value of the key in priority queue p, and where $\mathrm{data}(\langle k, d\rangle) = d$ and $\pi_1(\langle i, ptr\rangle) = i$. Thus, the following procedure makeSC ("make Strongly Concurrent") loops until the heads of the non-empty queues are strongly concurrent:

```
procedure makeSC()
    while $\neg\mathrm{SC}(p_1, p_2)$
        $\langle k, i\rangle$ := extractMin($p_1$);
        remove record for $i$ from $p_2$;
        removeHead($q_i$);
        if $\neg$empty($q_i$) then
            add records for $i$ to $p_1$ and $p_2$;
        endif
    endwhile
```

The optimized algorithm for detecting **Inst** appears in Figure 3, where head2(q) returns the second element of a queue q. When a SCGS g is found, if g does not satisfy Φ, then the algorithm starts searching for the next SCGS by advancing some process j such that this advance yields a CGS (*i.e.*, an element of $CGS^{\overset{e}{\twoheadrightarrow}}$). If at first no process can be so advanced (*e.g.*, if each queue q_i contains only one element), then the algorithm waits for more local states to be reported. It follows from the definitions of $CGS^{\overset{e}{\twoheadrightarrow}}$ and $CGS^{\overset{e}{\rightarrow}}$ that if some process can be so advanced, then a process j such that $C_1(\mathcal{S}(\mathrm{head2}(q_j)))$ is minimal can be so advanced. Thus, by maintaining a priority queue p_3 with key $C_1(\mathcal{S}(\mathrm{head2}(q_i)))$, a candidate process to advance can be found in constant time. We can determine in constant time whether advancing this candidate yields a CGS, using a test similar to (8), but with p_1 and p_2 replaced with appropriate priority queues. This requires maintaining an additional priority queue.

We analyze the worst-case time complexity of this algorithm by summing the times spent inside and outside of makeSC. Each iteration of the **while** loop in makeSC takes $O(\log N)$ time (because each operation on priority queues takes $O(\log N)$ time) and removes one local state. The computation contains $O(NE)$ local states, so the total time spent inside makeSC is $O((N \log N)E)$. The total time spent in the code outside makeSC is also $O((N \log N)E)$, since there are $O(NE)$ SCGSs (this is a corollary of Theorem 6), and each local state is considered at most once and at constant cost in the **wait** statement. Thus, the worst-case time complexity of the algorithm is $O((N \log N)E)$.

6 Sample Application: Debugging Coherence Protocols

Coherence of shared data is a central issue in many distributed systems, such as distributed file systems, distributed shared memory, and distributed databases. A typical invariant maintained by a coherence protocol is: if one machine has a copy of a data item in write mode, then no other machine has a valid copy of that

```
On receiving $x$ from process $i$:
    append($q_i, x$);
    if head($q_i$) = $x$ then
        add records for $i$ to $p_1$ and $p_2$;
        $found$ := true;
        while $found$
            makeSC();
            if ($\exists\, i$ : empty($q_i$)) then
                $found$ := false
            else (* found a SCGS *)
                $g$ := the global state ($\lambda i.$ head($q_i$));
                if $g$ satisfies $\Phi$ then
                    report $\mathbf{Inst}\,\Phi$ and exit
                else
                    wait until there exists $j$ such that $g[j \mapsto$ head2($q_j$)$]$ is in $CGS^{\overset{e}{\twoheadrightarrow}}(c)$;
                    remove records for $j$ from $p_1$ and $p_2$;
                    removeHead($q_j$);
                    add records for $j$ to $p_1$ and $p_2$
                endif
            endif
        endwhile
    endif
```

Fig. 3. Algorithm for detecting $\mathbf{Inst}\,\Phi$.

data item. Let *cohrnt* denote this predicate. As part of testing and debugging a coherence protocol, one might issue a warning if $\mathbf{Poss_T}\,\neg cohrnt$ is detected and report an error if $\mathbf{Def_T}\,\neg cohrnt$ is detected. A computationally cheaper but less informative alternative is to monitor only $\mathbf{Inst}\,\neg cohrnt$ and report an error if it is detected. In either case, if on-line detection would cause an unacceptable probe effect, then the probe effect can be reduced by logging timestamped local states locally and using the detection algorithms off-line.

A detection algorithm based on happened-before could be used instead, if the system can be modified to maintain vector clocks (or is unusual and maintains them already). However, if the coherence protocol uses timers—for example, if leases are used instead of locks—then time acts as a hidden channel [BM93] (*i.e.*, a means of communication other than messages), so detection based on happened-before might yield less precise results. For example, expiration of a lease and granting of another lease to a different machine need not be related by happened-before, so $\mathbf{Poss}\,\neg cohrnt$ may be detected, even though coherence was maintained and $\mathbf{Poss_T}\,\neg cohrnt$ would not be detected.

7 Related and Future Work

Marzullo and Neiger [MN91] define two detection modalities for partially-synchronous systems. In the notation of this paper, those modalities are $\mathbf{Poss}^{\overset{e}{\rightarrow}_{MN}}$ and $\mathbf{Def}^{\overset{e}{\rightarrow}_{MN}}$, where

$$e \overset{e}{\rightarrow}_{MN} e' \;\triangleq\; e \overset{e}{\twoheadrightarrow} e' \,\vee\, e \overset{e}{\rightarrow}_{hb} e'. \qquad (12)$$

Combining logical and real-time orderings in this way exploits more information about the computation but requires that the system maintain vector clocks. In [MN91], there is no discussion of an event ordering analogous to $\xrightarrow{e}$ or a modality analogous to **Inst**. Also, [MN91] considers only systems in which all clocks are always synchronized within a fixed offset ϵ, while our framework accommodates varying quality of synchronization.

Veríssimo [Ver93] discusses the uncertainty in event orderings caused by the granularity[6] and imperfect synchronization of digital real-time clocks, analyzes the conditions under which this uncertainty is significant for an application, and describes a synchronization technique, suitable for certain applications, that masks this uncertainty. However, [Ver93] does not aim for a general approach to detecting global predicates in the presence of this uncertainty.

This paper proposes a foundation for detection of global predicate in systems with approximately-synchronized real-time clocks. One direction for future work is to implement and gain experience with the detection algorithms. Another is to study efficient detection of global predicates that depend explicitly on time.

References

[BM93] Ö. Babaoğlu and K. Marzullo. Consistent global states of distributed systems: Fundamental concepts and mechanisms. In Sape Mullender, editor, *Distributed Systems*, ch. 5, pages 97–145. Addison Wesley, 2nd ed., 1993.

[CBDGF95] B. Charron-Bost, C. Delporte-Gallet, and H. Fauconnier. Local and temporal predicates in distributed systems. *ACM Trans. on Programming Languages and Systems*, 17(1):157–179, January 1995.

[CLR90] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* MIT Press and McGraw-Hill, 1990.

[CM91] R. Cooper and K. Marzullo. Consistent detection of global predicates. In *Proc. ACM/ONR Workshop on Parallel and Distributed Debugging*, 1991. Appeared as ACM SIGPLAN Notices 26(12):167-174, December 1991.

[DJR93] C. Diehl, C. Jard, and J.-X. Rampon. Reachability analysis on distributed executions. In J.-P. Jouannaud and M.-C. Gaudel, editors, *TAPSOFT '93: Theory and Practice of Software Development*, vol. 668 of *Lecture Notes in Computer Science*, pages 629–643. Springer, 1993.

[FR94] E. Fromentin and M. Raynal. Inevitable global states: a concept to detect unstable properties of distributed computations in an observer independent way. In *Proc. 6th IEEE Symposium on Parallel and Distributed Processing*, 1994.

[FR95] E. Fromentin and M. Raynal. Characterizing and detecting the set of global states seen by all observers of a distributed computation. In *Proc. IEEE 15th Int'l. Conference on Distributed Computing Systems*, 1995.

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

[6] Our framework accommodates the granularity of digital clocks by using $\leq$ instead of $<$ in SC1 and SC2.

[GW96] V. K. Garg and B. Waldecker. Detection of strong unstable predicates in distributed programs. *IEEE Trans. on Parallel and Distributed Systems*, 7(12):1323–1333, 1996.

[JMN95] R. Jegou, R. Medina, and L. Nourine. Linear space algorithm for on-line detection of global predicates. In J. Desel, editor, *Proc. Int'l. Workshop on Structures in Concurrency Theory (STRICT '95)*. Springer, 1995.

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–564, 1978.

[Mil91] D. L. Mills. Internet time synchronization: the Network Time Protocol. *IEEE Trans. Communications*, 39(10):1482–1493, October 1991.

[Mil95] D. L. Mills. Improved algorithms for synchronizing computer network clocks. *IEEE/ACM Transactions on Networking*, 3(3):245–254, June 1995.

[MN91] K. Marzullo and G. Neiger. Detection of global state predicates. In *Proc. 5th Int'l. Workshop on Distributed Algorithms (WDAG '91)*, vol. 579 of *Lecture Notes in Computer Science*, pages 254–272. Springer, 1991.

[SM94] R. Schwarz and F. Mattern. Detecting causal relationships in distributed computations: In search of the holy grail. *Distributed Computing*, 7(3):149–174, 1994.

[SS95] S. D. Stoller and F. B. Schneider. Faster possibility detection by combining two approaches. In J.-M. Hélary and M. Raynal, editors, *Proc. 9th Int'l. Workshop on Distributed Algorithms (WDAG '95)*, vol. 972 of *Lecture Notes in Computer Science*, pages 318–332. Springer, 1995.

[Tan95] A. S. Tanenbaum. *Distributed Operating Systems*. Prentice–Hall, 1995.

[Ver93] P. Veríssimo. Real-time communication. In Sape Mullender, editor, *Distributed Systems*, ch. 17, pages 447–490. Addison Wesley, 2nd ed., 1993.

Appendix

Proof of Theorem 3. It follows immediately from the definitions that $\overset{e}{\twoheadrightarrow}$ is process-wise-total. We need to show that $\overset{e}{\twoheadrightarrow}$ is irreflexive, acyclic, and transitive. Irreflexivity is obvious. For transitivity, we suppose $e \overset{e}{\twoheadrightarrow} e'$ and $e' \overset{e}{\twoheadrightarrow} e''$, and show $e \overset{e}{\twoheadrightarrow} e''$. First consider the case $pr(e) = pr(e'')$. If $pr(e') = pr(e)$, then the desired result follows from transitivity of "occurred before". Of $pr(e') \neq pr(e)$, then using SC1, the hypothesis $e \overset{e}{\twoheadrightarrow} e'$, SC1 again, and finally the hypothesis $e' \overset{e}{\twoheadrightarrow} e''$, we have the chain of inequalities $C_1(e) \leq C_2(e) < C_1(e') \leq C_2(e') < C_1(e'')$, so $C_1(e) < C_1(e'')$, so by SC2, e occurred before e''. Next consider the case $pr(e) \neq pr(e'')$. Note that $\neg(pr(e) = pr(e') \wedge pr(e') = pr(e''))$. If $pr(e') \neq pr(e)$, then it is easy to show that $C_2(e) < C_1(e') \leq C_1(e'')$, so $C_2(e) < C_1(e'')$, as desired. If $pr(e') \neq pr(e'')$, then it is easy to show that $C_2(e) \leq C_2(e') < C_1(e'')$, so $C_2(e) < C_1(e'')$, as desired.

Given transitivity, to conclude acyclicity, it suffices to show that there are no cycles of size 2. We suppose $e \overset{e}{\twoheadrightarrow} e'$ and $e' \overset{e}{\twoheadrightarrow} e$, and derive a contradiction. If $pr(e) = pr(e')$, then the fact that "occurred before" is a total order on the events of each process yields the desired contradiction. If $pr(e) \neq pr(e')$, then using SC1, the hypothesis $e \overset{e}{\twoheadrightarrow} e'$, SC1 again, and finally the hypothesis $e' \overset{e}{\twoheadrightarrow} e$, we obtain the chain of inequalities $C_1(e) \leq C_2(e) < C_1(e') \leq C_2(e') < C_1(e)$, which implies $C_1(e) < C_1(e)$, a contradiction. □

Proof of Theorem 4. Suppose not, *i.e.*, suppose there exist a computation c, global states g and g' in $CGS^{\overset{e}{\to}}(c)$, and processes i and j such that $g(i) \overset{e}{\to} g'(i)$ and $g'(j) \overset{e}{\to} g(j)$. By definition of $CGS^{\overset{e}{\to}}(c)$, $\neg(g(i) \overset{e}{\to} g(j))$, so $C_2(\mathcal{S}(g(j))) < C_1(\mathcal{T}(g(i)))$. By hypothesis, $g(i) \overset{e}{\to} g'(i)$, so $C_1(\mathcal{T}(g(i))) < C_2(\mathcal{S}(g'(i)))$, so by transitivity, $C_2(\mathcal{S}(g(j))) < C_2(\mathcal{S}(g'(i)))$. By definition of $CGS^{\overset{e}{\to}}(c)$, $\neg(g'(j) \overset{e}{\to} g'(i))$, so $C_2(\mathcal{S}(g'(i))) < C_1(\mathcal{T}(g'(j)))$, so by transitivity, $C_2(\mathcal{S}(g(j))) < C_1(\mathcal{T}(g'(j)))$. By hypothesis, $g'(j) \overset{e}{\to} g(j)$, so $C_1(\mathcal{T}(g'(j))) < C_2(\mathcal{S}(g(j)))$, so by transitivity, $C_2(\mathcal{S}(g(j))) < C_2(\mathcal{S}(g(j)))$, which is a contradiction. □

Proof sketch of Theorem 5. Consider an computation in which, at approximately the same time, event e_1 occurs on process 1 and events e_2 and e'_2 occur in rapid succession on process 2. If $C_2(e_1) - C_1(e_1)$, $C_2(e_2) - C_1(e_2)$, and $C_2(e'_2) - C_1(e'_2)$ are large relative to the separation (in time) between these events, then none of the actual orderings between e_1 and e_2 or between e_1 and e'_2 can be determined from the timestamps, so $e'_2 \overset{e}{\to} e_1$ and $e_1 \overset{e}{\to} e_2$. Since also $e_2 \overset{e}{\to} e'_2$, $\overset{e}{\to}$ contains a cycle and therefore is not a partial ordering.

Proof of Theorem 6. It suffices to show that a global state g is in $CGS^{\overset{e}{\to}}(c)$ iff it is contained in every maximal path of $CGS^{\overset{e}{\twoheadrightarrow}}(c)$. The proof of this is based on a result of Fromentin and Raynal. Recast in our notation, Theorem IGS of [FR94] (or Theorem C of [FR95]) states that a global state g is contained in every maximal path of $\langle CGS^{\overset{e}{\to}_{hb}}(c), \preceq_G \rangle$ iff $(\forall i, j : \mathcal{S}(g(i)) \overset{e}{\to}_{hb} \mathcal{T}(g(j)) \vee g(i) = \mathit{last}(c(i)))$, where *last* returns the last element of a sequence. A closely analogous proof shows that a global state g is contained in every maximal path of $\langle CGS^{\overset{e}{\twoheadrightarrow}}(c), \preceq_G \rangle$ iff $(\forall i, j : \mathcal{S}(g(i)) \overset{e}{\twoheadrightarrow} \mathcal{T}(g(j)))$, which by definition of $\overset{e}{\twoheadrightarrow}$ is equivalent to

$$(\forall i, j : i \neq j \Rightarrow C_2(\mathcal{S}(g(i))) < C_1(\mathcal{T}(g(j)))). \tag{13}$$

The only significant difference involves the treatment of the last local state of each process. Informally, the disjunct $g(i) = \mathit{last}(c(i))$ is needed in Fromentin and Raynal's analysis based on happened-before because, by a peculiarity of the definitions, the global state g_f containing the last local state of each process appears in every maximal path of $\langle CGS^{\overset{e}{\to}_{hb}}(c), \preceq_G \rangle$, even though the system might not have passed through g_f in real-time, since the processes might have terminated at different times. This peculiarity is absent from our analysis based on real-time timestamps—specifically, g_f appears in every maximal path of $\langle CGS^{\overset{e}{\twoheadrightarrow}}(c), \preceq_G \rangle$ iff the system necessarily passed through g_f in real-time—so (13) does not need a disjunct dealing specially with the last local state of each process. Expanding the definition of $CGS^{\overset{e}{\to}}(c)$ and simplifying yields (13). □

This article was processed using the LaTeX macro package with LLNCS style