Causality-Based Predicate Detection across Space and Time

Punit Chandra and Ajay D. Kshemkalyani, *Senior Member*, *IEEE*

Abstract—This paper presents event stream-based online algorithms that fuse the data reported from processes to detect causality-based predicates of interest. The proposed algorithms have the following features. 1) The algorithms are based on logical time, which is useful to detect "cause and effect" relationships in an execution. 2) The algorithms detect properties that can be specified using predicates under a rich palette of time modalities. Specifically, for a conjunctive predicate ϕ, the algorithms can detect the *exact fine-grained time modalities* between each pair of intervals, one interval at each process, with low space, time, and message complexities. The main idea used to design the algorithms is that any "cause and effect" interaction can be decomposed as a collection of interactions between *pairs of system components*. The detection algorithms, which leverage the *pairwise interaction* among the processes, incur a low overhead and are, hence, highly scalable. The paper then shows how the algorithms can deal with mobility in mobile ad hoc networks.

Index Terms—Predicates, event streams, causality, data fusion, time, space-time, mobility, ad hoc network, intervals, monitoring.

1 Introduction

Event-based data streams represent relevant state changes that occur at the various processes that are monitored in a distributed system. The paradigm of analyzing event streams using *data fusion* to detect predicates of interest to various applications is rapidly gaining importance. This paper proposes online algorithms for detecting causality-based predicates and behavioral patterns using event streams which are generated by potentially mobile processes across the system.

Stable global predicates (once a stable predicate becomes true, it remains true) can be detected using techniques such as snapshot algorithms [9]. However, unstable global predicates are difficult to detect. This is because an instantaneous observation across various locations is not possible due to the lack of perfectly synchronized clocks. Furthermore, the asynchronous nature of the distributed system—caused by unpredictable propagation delays and CPU loads, unknown scheduling policies, and mobility—leads to many interleavings of the events and different observations of their order. Hence, it is important to have a framework to *specify* various temporal modalities on the predicates on the various distributed variables and to *monitor* these predicates.

Two modalities, $Possibly(\phi)$ and $Definitely(\phi)$, for the satisfaction of a global predicate ϕ in a distributed execution were defined by Cooper and Marzullo [12] and are widely accepted [11]. Informally, $Possibly(\phi)/Definitely(\phi)$ is true if some/every observation of an execution will pass through a state in which ϕ is true. These two modalities are based on the *potential causality* or the "*happens before*" relation in distributed executions, which was defined by Lamport [24].

The formalism and axiom system given in [19] identified a complete, *orthogonal* set $\Re$ of 40 fine-grained temporal *relations* or *interaction types* (i.e., *modalities*) on time *interval pairs*, based on the potential causality relation in a distributed execution. Any two intervals at two processes are related by one and only one relation in $\Re$ and no relation in $\Re$ can be expressed in terms of others in $\Re$. It has been shown [20] that this formalism provides much more expressive power than the *Possibly* and *Definitely* modalities and a mapping from $\Re$ to the $Possibly$ and $Definitely$ modalities has been given.

A *conjunctive* predicate is of the form $\bigwedge_i \phi_i$, where ϕ_i is any predicate defined on variables local to process P_i. (An example is: $x_i = 5 \bigwedge y_j > 2$.) The local durations in which ϕ_i is true naturally identify intervals of interest at process P_i in an execution. We model such intervals as the "meta-events" that give rise to the "event stream" generated by each process. Information about the reported intervals is "fused" or correlated and examined in one of the following ways:

- To *detect* global states of the execution that satisfy a *given* predicate.
- To analyze causal relations between intervals to "data mine" patterns in the behavior.

The earliest global state in which a conjunctive predicate holds is well-defined due to the lattice structure of the set of global states [28]. We show that, for a conjunctive predicate ϕ, $Possibly(\phi)$ and $Definitely(\phi)$ can be detected along with the added information of the *exact interaction type* from $\Re$ between each pair of intervals, one interval at each process. This provides flexibility and power to monitor event streams generated from different mobile processes. The time, space, and message complexities of the proposed online, detection algorithms (Algorithms $Fine\_Poss$ and $Fine\_Def$) to detect $Possibly$ and $Definitely$ in terms of the

• *The authors are with the Computer Science Department, 851 S. Morgan St., University of Illinois at Chicago, Chicago, IL 60607. E-mail: {pchandra, ajayk}@cs.uic.edu.*

Manuscript received 27 May 2004; revised 14 Mar. 2005; accepted 15 July 2005; published online 16 Sept. 2005.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0222-0705.

TABLE 1
Comparison of Space, Message, and Time Complexities

	Avg. time complexity at P_0	Total number of messages	Space at P_0 (= total msg. space)	Avg. space at P_i, $i \in [1,n]$
$Fine\_Rel$	$O(n^2M)$ or $O(n[min(np, 2m_s + 2m_r)])$	$O(min(np, 2m_s + 2m_r))$	$O(min[(4n-2)np, 6nm_s + 4nm_r])$	$O(n)$
$Fine\_Poss$, $Fine\_Def$	$O(n^2M)$ or $O(n[min(np, 2m_s + 2m_r)])$	$O(min(np, 2m_s + 2m_r))$	$O(min[(4n-2)np, 6nm_s + 4nm_r])$	$O(n)$
$All\_Pairs\_Fine\_Rel$	$O(n^2p^2)$	$O(np)$	$O(n^2p)$	$O(n)$
$Possibly$ [15], $Definitely$ [16]	$O(n^2M)$ or $O(nm_r)$	$O(m_r)$	$O(n^2M)$ or $O(nm_r)$	$O(n)$

n = number of processes, M = maximum queue length at P_0, p = maximum number of intervals occurring at any process, m_s = number of messages sent by all the processes, m_r = number of messages received by all the processes. Note: 1) $p \geq M$ as all the intervals may not be sent to P_0. 2) $m_s = m_r$ for unicasts.

fine-grained modalities per pair of processes are the *same as* those of the earlier online algorithms [15], [16] that can detect *only* whether the $Possibly$ and $Definitely$ modalities hold. In addition, the proposed algorithms work in mobile ad hoc networks. $Fine\_Rel$, which is an intermediate problem we need to solve, is addressed first. $Fine\_Rel$ is important in its own right because it determines a global state such that a prespecified fine-grained modality for each pair of processes is satisfied. We also consider the problem $All\_Pairs\_Fine\_Rel$ to detect the fine-grained relation between each pair of intervals from different processes. The output can be useful in analyzing causal relations and mining behavioral patterns in the execution.

We now formally define the problems for which we design efficient online algorithms that fuse information from the event streams provided by (potentially) mobile processes/nodes.

Problem $Fine\_Rel$ **statement.** Given a relation $r_{i,j}$ from $\Re$ for each pair of processes P_i and P_j, identify the intervals (if they exist), one from each process, such that each relation $r_{i,j}$ is satisfied for the (P_i, P_j) pair.

Problem $Fine\_Poss$ **statement.** For a conjunctive predicate ϕ, determine online if $Possibly(\phi)$ is true. If true, identify the fine-grained pairwise interaction $r_{i,j}$ from $\Re$ between each pair of processes (P_i, P_j) when $Possibly(\phi)$ first becomes true.

Problem $Fine\_Def$ **statement.** For a conjunctive predicate ϕ, determine online if $Definitely(\phi)$ is true. If true, identify the fine-grained pairwise interaction $r_{i,j}$ from $\Re$ between each pair of processes (P_i, P_j) when $Definitely(\phi)$ first becomes true.

Problem $All\_Pairs\_Fine\_Rel$ **statement.** Devise an online algorithm to determine which relation from $\Re$ holds between each pair of intervals from different processes.

The proposed algorithms detect "cause-effect" relationships from the information in event streams, using predicates under a *rich palette of temporal specifications*. The performance of the algorithms is compared in Table 1. P_0 is the data fusion server that processes the event streams. The metrics are the average time complexity at P_0, the total space complexity at P_0, the cumulative size of all the messages sent to P_0, the network-wide count of the messages sent by the processes to P_0, and the average space at each process P_i. The parameters n, M, p, m_s, and m_r are explained in the legend.

For $Fine\_Rel$, $Fine\_Poss$, and $Fine\_Def$, all the measures at P_0 are either linear or quadratic in n, the size of the ad hoc network. This indicates high scalability of the algorithms. More importantly, the size of the space requirement at each process P_i ($i \in [1,n]$), which is both space and energy constrained in a mobile ad hoc network, is linear in n, the number of mobile processes. This makes the algorithm easily implementable. The overhead can be further lowered by using a hierarchical structuring of the ad hoc network into clusters, with the data fusion server at its root. Known algorithms for managing various issues due to migration and mobility [2], [3], [4], [26], [30], [32] can be used in conjunction to maintain the hierarchical cluster organization.

The paper is organized as follows: Section 2 gives the background and objectives. Section 3 gives the framework and data structures. Sections 4, 5, and 6 give the online detection algorithms. Section 7 shows how the algorithms can run on mobile ad hoc networks. Section 8 gives the concluding remarks and discusses further research challenges.

2 Model and Background

2.1 System Model

The system model is broad enough to accommodate not just traditional distributed computing environments, but also mobile ad hoc networks. The model assumes a loosely coupled ad hoc asynchronous message-passing system in which any two processes belonging to the process set $N = \{P_1, P_2, \ldots, P_n\}$ can communicate over logical channels. For a wireless communication system, a physical channel exists from P_i to P_j if and only if P_j is within P_i's range; a logical channel is a sequence of physical channels representing a multihop path. Explicit mobility is permitted in the model and both processes and nodes can migrate within the ad hoc network. The only requirement is that each process be able to send its gathered data eventually and asynchronously (via any routes) in a FIFO stream to a system-wide known data fusion server. Known techniques for managing mobility and clustering can be used within a mobile ad hoc

network. In the remainder of this paper, the term "process" will be used to denote a mobile process. We assume that a single process runs at each node; thus, node mobility and process mobility also become synonymous.

E_i is the linearly ordered set of events executed by process P_i in an execution. An event executed by P_i is denoted e_i. An execution is modeled as $(E, \prec)$, where $E = \bigcup_i E_i$ is the set of all the events and $\prec$ is the *potential causality* or the "*happens before*" relation, defined as the transitive closure of the local ordering relation on each E_i and the ordering imposed by message send events and message receive events [24]. A *cut* C is a subset of E such that if $e_i \in C$ then $(\forall e_i')\ e_i' \prec e_i \Longrightarrow e_i' \in C$. Thus, the events of a cut are downward-closed within each E_i. A *consistent cut* is a downward-closed subset of E. Only all downward-closed subsets of E preserve causality and denote correctly observable execution prefixes.

A variable x local to process P_i is denoted as x_i. Given a networkwide predicate on the variables, the *intervals* of interest at each process are the durations during which the local predicate is true. Such an interval at process P_i is identified by the (totally ordered) corresponding adjacent events within E_i for which the local predicate is true. Intervals are denoted by capitals X, Y, and Z. For event e, there are two special consistent cuts $\downarrow e$ and $e \uparrow$. $\downarrow e$ is the maximal set of events that happen before e. $e \uparrow$ is the set of all events up to and including the earliest events at each process for which e happens before the events.

Definition 1 (Past and future cuts). *The cut $\downarrow e$ is defined as $\{e' \mid e' \prec e\}$. The cut $e \uparrow$ is defined as*

$$\{e' \mid e' \not\succeq e\} \bigcup \{e_i\ (1 \le i \le n) \mid (e_i \succeq e) \bigwedge (\forall e_i'\ \ for\ which\ e_i' \prec e_i, e_i' \not\succeq e)\}.$$

The system state after the events in a cut is a global state; if the cut is consistent, the corresponding system state is termed a *consistent global state* and denotes a meaningful observation of a global state [9]. Each interval can be viewed as defining an event of coarser granularity at that process, as far as the local predicate of interest is concerned. Such higher-level events [19], one from each process, can also be used to identify a global state [9].

We assume that vector clocks are available [11], [14], [28]—the vector clock V has the property that $e \prec f \Longleftrightarrow V(e) < V(f)$. Such vector clocks provide *logical time* in the system. Each process P_i maintains a vector clock V_i of size n. The clock operation is described later in Section 3.2. Logical time and vector clocks evolved as a substitute for physical time for causality-based applications [33], [34]. We use logical vector clocks because they offer the following advantages over synchronized (scalar) physical clocks:

1. Vector clocks inherently capture the partial order $(E, \prec)$ and the causality relation in the execution by correlating send and receive events across the processes [14], [22], [28]. Specifically, causal relationships get manifested in network messages and clock values.
2. Vector clocks can inherently capture and represent the knowledge of the states (i.e., of a relevant property at the various states) of other processes [22]. At any event, the latest progress that can be inferred about other processes is available.

These advantages help in fusing causality-based event information and reduce the postprocessing; inferencing of causality-based properties can be easily performed using the axiom system [19]. Physical clocks define a linear order on the events and do not provide a handle to represent the progress known to be made by other processes. Further, interprocess dependencies [25] are not captured. Using physical clocks, if event e at P_i has a smaller timestamp than event f at P_j, the only definitive conclusion is that there is no causal dependency from f to e, i.e., $f \not\prec e$. Nothing can be inferred about $e \prec f$ [22], [33]. Thus, physical clocks —whether perfect or synchronized via GPS [27], NTP [29], or any of the many clock synchronization protocols for wired as well as wireless media, such as those surveyed in [13], [36]—do not offer the mechanism for dealing with causality relations.

2.2 Background

2.2.1 $Possibly/Definitely$ Temporal Modalities

Specifying predicates on the system state provides an important handle to specify and detect the behavior of a system. For unstable predicates, the condition encoded by the predicate may not persist long enough for it to be true when the predicate is evaluated. Furthermore, due to the inherent limitations of observing a distributed system, even if a predicate is found to be true by a central monitor, it may not have ever held during the actual execution. Specifically, global snapshot protocols are not useful to detect unstable predicates—even if the predicate is detected, it may never have held and the predicate may have held even if it is not detected. Cooper and Marzullo [12] therefore proposed two modalities that apply to the entire distributed execution.

- $Possibly(\phi)$: There exists a consistent observation of the execution such that predicate ϕ 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 predicate ϕ holds.

2.2.2 Conjunctive Predicates

Cooper and Marzullo also proposed an online centralized algorithm to detect $Possibly(\phi)$ and $Definitely(\phi)$ for an arbitrary predicate ϕ. The algorithm works by building a lattice of global states. Although it detects generalized global predicates, the complexity of the algorithm is c^n, where c is the maximum number of events on any process and n is the number of processes. To reduce the complexity of the algorithm, researchers have focused on the important class of *conjunctive global predicates*. A conjunctive predicate $\phi = \wedge_i \phi_i$ can be expressed as a conjunction of various local predicates ϕ_i, where ϕ_i is defined on variables that are local to process P_i. In this paper, we consider such conjunctive predicates. Garg and Waldecker [15], [16] presented centralized algorithms to detect $Possibly(\phi)$ and $Definitely(\phi)$ for conjunctive predicates, with message space, storage, and time

TABLE 2
Dependent Relations for Interactions between Intervals X and Y [19]

Relation r	Expression for $r(X,Y)$	Test for $r(X,Y)$
R1	$\forall x \in X \forall y \in Y, x \prec y$	$V_y^-[x] > V_x^+[x]$
R2	$\forall x \in X \exists y \in Y, x \prec y$	$V_y^+[x] > V_x^+[x]$
R3	$\exists x \in X \forall y \in Y, x \prec y$	$V_y^-[x] > V_x^-[x]$
R4	$\exists x \in X \exists y \in Y, x \prec y$	$V_y^+[x] > V_x^-[x]$
S1	$\exists x \in X \forall y \in Y, x \not\preceq y \bigwedge y \not\preceq x$	**if** $V_y^-[y] \not< V_x^-[y] \bigwedge V_y^+[x] \not> V_x^+[x]$ **then** $(\exists x^0 \in X$: $V_y^-[y] \not\leq V_x^{x^0}[y] \wedge V_x^{x^0}[x] \not\leq V_y^+[x])$ **else** $false$
S2	$\exists x_1, x_2 \in X \exists y \in Y, x_1 \prec y \prec x_2$	**if** $V_y^+[x] > V_x^-[x] \bigwedge V_y^-[y] < V_x^+[y]$ **then** $(\exists y^0 \in Y : V_x^+[y] \not< V_y^{y^0}[y] \wedge V_y^{y^0}[x] \not< V_x^-[x])$ **else** $false$

Tests for the relations [20] are given in the third column and are explained in Section 3.

TABLE 3
The 40 Orthogonal Relations in $\Re$ [19]

Orthogonal relations (interaction types)	Dependent relations $r(X,Y)$						Dependent relations $r(Y,X)$					
	R1	R2	R3	R4	S1	S2	R1	R2	R3	R4	S1	S2
$IA\ (=IQ^{-1})$	1	1	1	1	0	0	0	0	0	0	0	0
$IB\ (=IR^{-1})$	0	1	1	1	0	0	0	0	0	0	0	0
$IC\ (=IV^{-1})$	0	0	1	1	1	0	0	0	0	0	0	0
$ID\ (=IX^{-1})$	0	0	1	1	1	1	0	1	0	1	0	0
$ID'\ (=IU^{-1})$	0	0	1	1	0	1	0	1	0	1	0	1
$IE\ (=IW^{-1})$	0	0	1	1	1	1	0	0	0	1	0	0
$IE'\ (=IT^{-1})$	0	0	1	1	0	1	0	0	0	1	0	1
$IF\ (=IS^{-1})$	0	1	1	1	0	1	0	0	0	1	0	1
$IG\ (=IG^{-1})$	0	0	0	0	1	0	0	0	0	0	1	0
$IH\ (=IK^{-1})$	0	0	0	1	1	0	0	0	0	0	1	0
$II\ (=IJ^{-1})$	0	1	0	1	0	0	0	0	0	0	1	0
$IL\ (=IO^{-1})$	0	0	0	1	1	1	0	1	0	1	0	0
$IL'\ (=IP^{-1})$	0	0	0	1	0	1	0	1	0	1	0	1
$IM\ (=IM^{-1})$	0	0	0	1	1	0	0	0	0	1	1	0
$IN\ (=IM'^{-1})$	0	0	0	1	1	1	0	0	0	1	0	0
$IN'\ (=IN'^{-1})$	0	0	0	1	0	1	0	0	0	1	0	1
$ID''\ (=(IUX)^{-1})$	0	0	1	1	0	1	0	1	0	1	0	0
$IE''\ (=(ITW)^{-1})$	0	0	1	1	0	1	0	0	0	1	0	0
$IL''\ (=(IOP)^{-1})$	0	0	0	1	0	1	0	1	0	1	0	0
$IM''\ (=(IMN)^{-1})$	0	0	0	1	0	0	0	0	0	1	1	0
$IN''\ (=(IMN')^{-1})$	0	0	0	1	0	1	0	0	0	1	0	0
$IMN''\ (=(IMN'')^{-1})$	0	0	0	1	0	0	0	0	0	1	0	0

The upper part gives the 29 relations assuming dense time. The lower part gives 11 additional relations if dense time is assumed.

complexity of $O(n^2m)$, where m is the maximum number of messages sent by any process. Distributed algorithms to detect $Possibly(\phi)$ and $Definitely(\phi)$ such as those in [18] and [7], respectively, have at least as much complexity.

2.2.3 Orthogonal Set of Pairwise Modalities

Table 2 gives six causality-based relations that were defined in [19] to capture the interaction between a pair of intervals. The relations R1-R4 and S1-S2 are expressed in terms of the quantifiers over intervals X and Y. Relations R1 (strong precedence), R2 (partially strong precedence), R3 (partially weak precedence), R4 (weak precedence) define *causality conditions*. S1 (independent events) and S2 (round-trip interaction) define *coupling conditions*.

Two models of time are assumed—*dense* and *nondense*. In the *dense time model*, an interval is defined over an infinite dense set E; between any two events in the interval, there is yet another event. Assuming dense time, there are 29 possible mutually exclusive interaction types between a pair of intervals [19], as given in the upper part of Table 3. The 29 interaction types are specified using Boolean vectors. The six relations R1-R4 and S1-S2 on (X,Y) and on (Y,X)

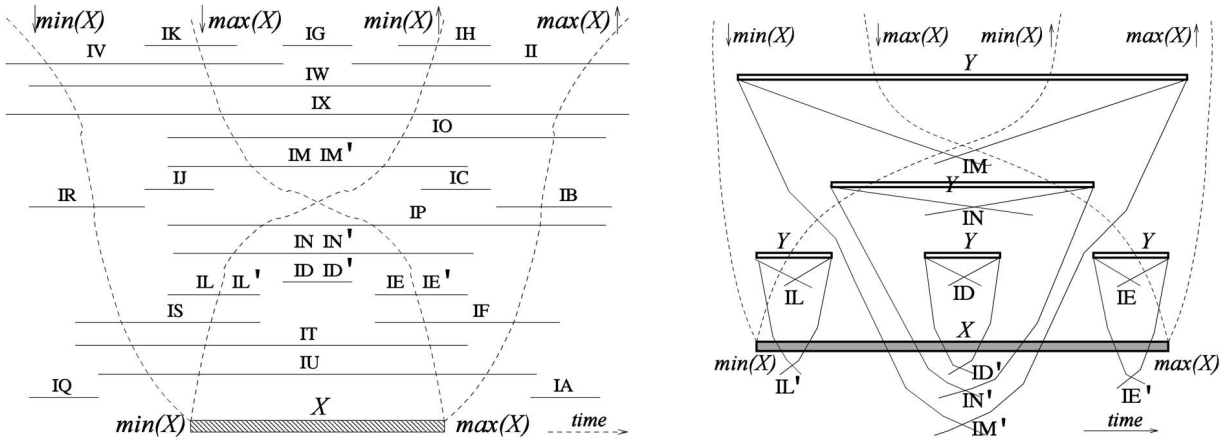

Fig. 1. Illustration of interaction types between intervals under the dense time model [19].

form a Boolean vector of length 12. Any two intervals at two processes are related by one and only one interaction type and no interaction type can be expressed in terms of the other interaction type—hence, the 29 interaction types form an *orthogonal* set of relations. Among these 29 interactions, there are 13 pairs of inverses and three are self-inverses. For example, $IH(X, Y)$ is equivalent to $IK(Y, X)$ as IH and IK are inverses of each other. Fig. 1 illustrates these interactions. Interval X is shown in a fixed position using a shaded rectangle, whereas multiple instances of interval Y, indicated using horizontal lines, are in different positions relative to X. $\min(X)$ and $\max(X)$ denote the least and greatest element of interval X, respectively. $\downarrow \min(X)$, $\downarrow \max(X)$, $\min(X) \uparrow$, and $\max(X) \uparrow$ denote consistent cuts. The interaction type of each position of Y relative to X is labeled by IA through IX (and their modifiers). Five positions of Y have two labels each. The distinction between the two interaction types represented in each of the five pairs of labels arises due to the $S1$ and $S2$ relations and is shown at the right.

The *nondense time model* allows 11 interaction types between a pair of intervals, defined in the lower part of Table 3, in addition to the 29 identified above. There are five pairs of inverses and one is its own inverse. These interactions are illustrated in [19], which also explains the significance of the nondense time model. The set of 40 orthogonal *interaction types*, *also* referred to interchangably as *modalities* or *relations*, henceforth is denoted $\Re$. Examples of how the $\Re$ modalities provide useful information that is not provided by the $Possibly/Definitely$ classification are given in [20].

2.2.4 Modalities for Global Predicates

Observe that, for any predicate ϕ, three orthogonal relational possibilities hold under the *Possibly/Definitely* classification: 1) $Definitely(\phi)$, 2) $\neg Definitely(\phi) \wedge Possibly(\phi)$, 3) $\neg Possibly(\phi)$. The 40 fine-grained orthogonal relational possibilities in $\Re$ have been mapped to the three orthogonal relational possibilities based on the *Possibly/Definitely* classification (see [20]). Table 4 shows this mapping, assuming that the conjunctive predicate ϕ is defined on two processes. When conjunctive predicate ϕ is defined over variables that are local to $n > 2$ processes, one can still express the three possibilities 1) $Definitely(\phi)$, 2) $\neg Definitely(\phi) \wedge Possibly(\phi)$, and 3) $\neg Possibly(\phi)$, in terms of the fine-grained 40 independent relations between C_2^n pairs of intervals. Not all $40^{C_2^n}$ combinations will be valid—the combinations have to satisfy the axiom system given in [19].

Theorem 1 [20]. *Consider a conjunctive predicate $\phi = \wedge_i \phi_i$. The following results are implicitly qualified over a set of intervals, containing one interval per process:*

- $Definitely(\phi)$ *holds if and only if* $\bigwedge_{(\forall i \in N)(\forall j \in N)} [Definitely(\phi_i \wedge \phi_j)]$.
- $\neg Definitely(\phi) \wedge Possibly(\phi)$ *holds if and only if*

$$(\exists i \in N)(\exists j \in N) \neg Definitely(\phi_i \wedge \phi_j) \bigwedge \left(\bigwedge_{(\forall i \in N)(\forall j \in N)} [Possibly(\phi_i \wedge \phi_j)] \right).$$

- $\neg Possibly(\phi)$ *holds if and only if*

$$(\exists i \in N)(\exists j \in N) \neg Possibly(\phi_i \wedge \phi_j).$$

Example. Let ϕ be $a_i = 8 \wedge b_j = 7 \wedge c_k = 2$. Let a_i, b_j, and c_k be 8, 7, 2, respectively, in intervals X_i, Y_j, and Z_k, respectively, and let $IC(X_i, Y_j)$, $IR(Y_j, Z_k)$, and $IH(Z_k, X_i)$ be true. See Fig. 2. Then, by Theorem 1, we have 1) $\neg Definitely(a_i = 8 \wedge b_j = 7)$ and $Possibly(a_i = 8 \wedge b_j = 7)$, 2) $\neg Definitely(b_j = 7 \wedge c_k = 2)$ and $Possibly(b_j = 7 \wedge c_k = 2)$, and 3) $\neg Definitely(a_i = 8 \wedge c_k = 2)$ and $Possibly(a_i = 8 \wedge c_k = 2)$. By Theorem 1, we have the modality $Possibly(\phi) \wedge \neg Definitely(\phi)$. Conversely, if $Possibly(\phi) \wedge \neg Definitely(\phi)$ is known in the classical course-grained classification, the fine-grained classification gives the added information: $IC(X_i, Y_j)$, $IR(Y_j, Z_k)$, and $IH(Z_k, X_i)$.

TABLE 4
Refinement Mapping [20]

$Definitely(\phi)$	$Possibly(\phi) \wedge \neg Definitely(\phi)$	$\neg Possibly(\phi)$
ID and IX ID′ and IU IE and IW IE′ and IT IF and IS IO and IL IP and IL′ IM IM′ and IN IN′	IB and IR IC and IV IG IH and IK II and IJ	IA and IQ
ID″ and IUX IE″ and ITW IL″ and IOP IN″ and IMN′	IM″ and IMN IMN″	

The upper part shows the 29 mappings under the dense time model. Under the nondense time model, the 11 additional mappings in the lower part also apply.

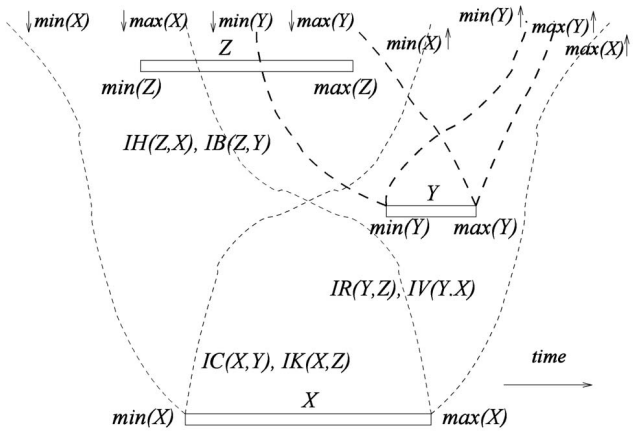

Fig. 2. Example to show fine-grained relations across $n = 3$ processes.

3 Framework and Data Structures

3.1 Framework

Given a conjunctive predicate, for each pair of intervals belonging to different processes, each of the 29 (40) possible independent relations in the dense (nondense) model of time can be tested for using the bit-patterns for the dependent relations, as given in Table 3. The tests for the relations $R1$, $R2$, $R3$, $R4$, $S1$, and $S2$ are given in the third column of Table 2 using vector timestamps [14], [28]. Recall that the interval at a process is used to identify the period when some local property (using which the predicate ϕ is defined) holds. V_i^- and V_i^+ denote the vector timestamps at process P_i at the start of an interval and the end of an interval, respectively. V_i^x denotes the vector timestamp of event x_i at P_i. Observe from Table 2 that, to test for R1, R2, R3, and R4, an overhead of $O(1)$ time complexity is needed if the timestamps of the start and the end of the intervals are known. However, for relations S1 and S2, if the tests are applied in a naive manner, then it is also necessary to know about all the events in each interval.

Processes $P_1, P_2, \ldots, P_n$ need to identify appropriate data, such as the timestamps of the start and end events of their local intervals and timestamps of other communication events within the intervals, to track locally. This information can be sent using the network asynchronously and without any timing requirements to a central data fusion server P_0. The only requirement is that, between any process and the data fusion server, the logical link must be FIFO. The data fusion server maintains queues $Q_1, Q_2, \ldots, Q_n$ for interval information from each of the processes. The server P_0 itself may be a more powerful computing device than the typical nodes deployed in the field. For each of the problems to be solved, the server runs an appropriately devised algorithm to process the interval information it receives in the queues. The algorithm logic has to detect "concurrent" pairwise interactions for each pair of intervals, considering only one interval from each process at a time as being a part of a potential solution. A challenge to solving $Fine\_Rel$ is to formulate the necessary and sufficient conditions to determine when to eliminate the received information about intervals from the queues in order to process the queues efficiently. These conditions are critical because intervals that have been determined as not forming part of a solution should get deleted from the queues at P_0 so that they are not considered in testing with other intervals. Efficient data structures also need to be identified carefully. It is important that the online algorithm has very low message, space, and time complexity, particularly keeping in mind the requirements of mobile networks—low power availability and low bandwidth utilization.

For $Fine\_Poss$ and $Fine\_Def$, a critical new challenge is to formulate different necessary and sufficient conditions (than for $Fine\_Rel$) to determine when to eliminate information about the intervals in the queues at P_0. This is because the deletion of the interval information cannot be done based on the examination of whether a single relation $r_{i,j}$ is satisfied for processes (P_i, P_j), as will be shown in Section 5.

We assume that interval X occurs at P_i and interval Y occurs at P_j. For any two intervals X and X' that occur at the same process, if $R1(X, X')$, then we say that X is a *predecessor* of X' and X' is a *successor* of X. Also, there are a maximum of p intervals at any process. The operations and data structures required by the algorithms to solve Problems $Fine\_Rel$, $Fine\_Poss$, $Fine\_Def$, and $All\_Pairs\_Fine\_Rel$ can be divided into two parts. The first part, common to all the algorithms, runs on each of the n processes P_1 to P_n and is given in this section. The second part of each algorithm runs on the data fusion process P_0 and is presented in later sections.

3.2 Clock and Log Operations

Each process P_i, where $1 \leq i \leq n$, maintains the following data structures:

- V_i : array $[1..n]$ of integer. This is the vector clock.
- I_i : array $[1..n]$ of integer. This is an *Interval Clock*, which tracks the latest intervals at processes. $I_i[j]$ is the timestamp $V_j[j]$ when ϕ_j last became true.

1. When an internal event or send event occurs at process P_i, $V_i[i] = V_i[i] + 1$.
2. Every message contains the vector clock and the *Interval Clock* of its send event.
3. When process P_i receives a message msg, then for j from 1 to n, do as follows.
   if $(j = i)$ **then** $V_i[i] = V_i[i] + 1$
   else $V_i[j] = \max(V_i[j], msg.V[j])$.
4. When an interval starts at P_i (local predicate ϕ_i becomes true), $I_i[i] = V_i[i]$.
5. When process P_i receives a message msg, then for j from 1 to n, do as follows.
   $I_i[j] = \max(I_i[j], msg.I[j])$.

Fig. 3. The vector clock and the *Interval Clock*.

type *Event_Interval* = **record**
  interval_id : integer;
  local_event: integer;
end

type *Process_Log* = **record**
  event_interval_queue: queue of *Event_Interval*;
end

type *Log* = **record**
  start: array[1..n] of integer;
  end: array[1..n] of integer;
  p_log: array[1..n] of *Process_Log*;
end

Start of an interval
  $Log_i.start = V_i^-$. //Store the timestamp V_i^- of the start of the interval.
On receiving a message during an interval
  if (change in I_i) **then** //Store local component of vector clock and *interval_id*
    for each k such that $I_i[k]$ was changed //which caused the change in I_i
      insert $(I_i[k], V_i[i])$ in $Log_i.p\_log[k].event\_interval\_queue$.
End of an interval
  $Log_i.end = V_i^+$ //Store the timestamp V_i^+ of the end of the interval.
  if (a receive or send event occurs between the start of the previous interval and the end of the current interval) **then** send Log_i to the central process.

Fig. 4. Data structures and operations to construct Log at P_i $(1 \leq i \leq n)$.

- Log_i: Contains the information to be sent to the central process.

Fig. 3 shows how to maintain the vector clock (Rules 1-3) [14], [28] and the *Interval Clock* (Rules 2, 4, 5) proposed in this paper. To maintain Log_i, the required data structures and operations are defined in Fig. 4. Log_i is constructed and sent to the central process using the protocol shown. The central process uses the *Log*s received to determine the relationship between two intervals.

3.3 Complexity Analysis at P_i $(1 \leq i \leq n)$

Theorem 2. *The average space complexity at P_i $(1 \leq i \leq n)$ is $6n - 2$ integers.*

Proof. Consider the construction algorithm for *Log* (Fig. 4). Each *Log* stores the start (V^-) and the end (V^+) of an interval, which requires $2n$ integers. Furthermore, an *Event_Interval* is added to the *Log* for every component of *Interval Clock* which is modified due to the receive of a message. As a change in a component of *Interval Clock* implies the start of a new interval on another process, the total number of times the component of *Interval Clock* can change is equal to the number of intervals on all the processes. Thus, the total number of *Event_Interval* which can be added to the *Log*s of a single process is $(n-1)p$. This takes $2(n-1)p$ integers per process. Thus, the total space for a single *Log* at a process can be up to $2n + 2(n-1)p$. However, the total space needed for *Log*s corresponding to all p intervals on a single process is $2(n-1)p + 2np$. This gives an average of $4n - 2$ integers per *Log* at a process. As only one *Log* exists at a time, the average space requirement at a process P_i $(1 \leq i \leq n)$ at any time is the sum of space required by vector clock, *Interval Clock*, and *Log*, which is $6n - 2$ integers. □

Theorem 3 (Message complexity).

- *The total number of messages sent by the processes to P_0 is $O(min(np, 2m_s + 2m_r))$.*
- *The total message space overhead is $O(min(4n^2p - 2np, (6m_s + 4m_r)n))$.*

Proof. The message complexity can be determined in two ways:

1. As one message is sent per interval, the number of messages is $O(p)$ for each P_i $(i \neq 0)$. This gives a

total message complexity of $O(np)$. On average, the size of each message is $4n-2$ as each message contains the Log. The total message space overhead for a particular process is the sum of all the Logs for that process, which was shown to be $4np-2p$. Hence, the total message space complexity is $4n^2p-2np=O(n^2p)$.

2. *An optimization of message size* can be used. The Log corresponding to an interval is sent to the central process only if the relationship between the interval and all other intervals (at other processes) is different from the relationship which its predecessor interval had with all the other intervals (at other processes). Two successive intervals, Y and Y', on process P_j will have the same relationship if no message is sent or received by P_j between the start of Y and the end of Y'. For each message exchanged between processes, a maximum of four interval Logs need to be sent to the central process because two successive intervals (Y and Y') will have different relationships if a receive or a send occurs between the start of Y and end of Y'. This makes it necessary to send two interval Logs for a send event and two for a receive event. Hence, if there are m_s number of send events and m_r number of receive events in the execution ($m_s=m_r$ for unicasts), then a total of $2m_s+2m_r$ intervals needs to be sent to the central process in $2m_s+2m_r$ control messages, while the total message space overhead is bounded by

$$2m_s.n+(2m_s+2m_r).2n=6m_sn+4m_rn.$$

The first term, $2m_s.n$, arises because, for every message sent, each other process eventually (due to transitive propagation of $Interval\ Clock$) may need to insert an $Event\_Interval$ tuple in its Log. This can generate an upper bound of $2nm_s$ overhead in Logs across the n processes. The second term, $(2m_s+2m_r).2n$, arises because the vector clock at the start and end of each interval is sent with each of the $2nm_s+2nm_r$ control messages.

Hence, the total number of control messages sent to P_0 and the total message space overhead is the lesser of when either all the intervals are sent or when four intervals are sent for each message exchanged. Thus, the total number of messages sent is $O(min(np,2m_s+2m_r))$ and the total message space overhead is $O(min(4n^2p-2np,(6m_s+4m_r)n))$. □

4 Algorithm $Fine\_Rel$ (Fine-Grained Relations)

To solve Problems *Fine_Poss* and *Fine_Def*, we first solve the intermediate problem *Fine_Rel*, which was defined in Section 1. Recall that the given relations $\{r_{i,j},\forall i,j\}$ must satisfy the axioms on $\Re$ [19] for a solution to potentially exist.

Algorithm Overview. The algorithm detects a set of intervals, one on each process, such that each pair of intervals satisfies the relationship specified for that pair of processes. If no such set of intervals exists, the algorithm does not return any interval set. The central process P_0 maintains n queues, one for Logs from each process, and determines which orthogonal relation holds between pairs of intervals. The queues are processed using "pruning," described below. If there exists an interval at the head of each queue and these intervals cannot be pruned, then these intervals satisfy $r_{i,j}\ \forall\ i,j$, where $i\neq j$ and $1\leq i,j\leq n$, and, hence, these intervals form a solution set.

The notion of pruning is developed by first defining the *prohibition function* and the *allows relation*.

Definition 2 (Prohibition function). *Function* $\mathcal{H}:\Re\rightarrow 2^{\Re}$ *is defined to be* $\mathcal{H}(r_{i,j})=\{R\in\Re\mid R\neq r_{i,j}\wedge\ if\ R(X,Y)\ is$ true $then\ r_{i,j}(X,Y')\ is$ false $for\ all\ Y'\ that\ succeed\ Y\ \}$.

Intuitively, for each $r_{i,j}\in\Re$, the *prohibition function* $\mathcal{H}(r_{i,j})$ is the set of all orthogonal relations R ($\neq r_{i,j}$) such that if $R(X,Y)$ is true, then $r_{i,j}(X,Y')$ can never be true for any successor Y' of Y. $\mathcal{H}(r_{i,j})$ is the set of relations that prohibit $r_{i,j}$ from being true in the future.

Two relations, R' and R'' (where $R'\neq R''$), in $\Re$ are related by the *allows relation* $\rightsquigarrow$ if the occurrence of $R'(X,Y)$ does not prohibit $R''(X,Y')$ for some successor Y' of Y.

Definition 3 ("allows" relation). $\rightsquigarrow$ *is a relation on* $\Re\times\Re$ *such that* $R'\rightsquigarrow R''$ *if 1)* $R'\neq R''$ *and 2) if* $R'(X,Y)$ *is* true, *then* $R''(X,Y')$ *can be* true *for some* Y' *that succeeds* Y.

Example. $IC\rightsquigarrow IB$ because 1) $IC\neq IB$ and 2) if $IC(X,Y)$ is true, then there is a possibility that $IB(X,Y')$ is also true, where Y' succeeds Y.

Lemma 1. *For* $R\neq r_{i,j}$, *if* $R\in\mathcal{H}(r_{i,j})$, *then* $R\not\rightsquigarrow r_{i,j}$, *else if* $R\notin\mathcal{H}(r_{i,j})$, *then* $R\rightsquigarrow r_{i,j}$.

Proof. If $R\in\mathcal{H}(r_{i,j})$, using Definition 2, it can be inferred that if $R(X,Y)$ is true, then $r_{i,j}$ is false for all Y' that succeed Y. This does not satisfy the second part of Definition 3. Hence, $R\not\rightsquigarrow r_{i,j}$. If $R\notin\mathcal{H}(r_{i,j})$ and $R\neq r_{i,j}$, it follows that if $R(X,Y)$ is true, then $r_{i,j}(X,Y')$ can be true for some Y' that succeeds Y. This satisfies Definition 3 and, hence, $R\rightsquigarrow r_{i,j}$. □

Table 5 gives $\mathcal{H}(r_{i,j})$ for each of the 40 interaction types in $\Re$. The table is constructed by analyzing each interaction pair in $\Re$.

Example. The third row gives $\mathcal{H}(r_{i,j})$ for IC and IV. From the second column,

$$\mathcal{H}(IC_{i,j})(X_i,Y_j)=\{IA,IB,IF,II,IP,IO,IUX,IOP\}.$$

Hence, for instance, if $IB(X_i,Y_j)$, then $IC(X_i,Y_j')$ will never hold for any successor Y_j' of Y_j. From the third column, $\mathcal{H}(IV_{i,j})(X_i,Y_j)=\Re\setminus\{IQ,IV\}$. Hence, for instance, if $IH(X_i,Y_j)$, then $IV(X_i,Y_j')$ will never hold for any successor Y_j' of Y_j.

We now state an important result between any two relations in $\Re$ that satisfy the "allows" relation and the existence of the "allows" relation between their respective inverses. Specifically, if R' allows R'', then Theorem 4 states that R'^{-1} necessarily does not allow relation R''^{-1}. The

TABLE 5
$\mathcal{H}(r_{i,j})$ for the 40 Independent Relations in $\Re$

Relation $r_{i,j} \in \Re$ (interaction type)	$\mathcal{H}(r_{i,j})(X_i, Y_j)$	$\mathcal{H}(r_{j,i})(Y_j, X_i)$
$IA\ (= IQ^{-1})$	ϕ	$\Re \setminus \{IQ\}$
$IB\ (= IR^{-1})$	$\{IA, IF, II, IP, IO, IU, IX, IUX, IOP\}$	$\Re \setminus \{IQ, IR\}$
$IC\ (= IV^{-1})$	$\{IA, IB, IF, II, IP, IO, IU, IX, IUX, IOP\}$	$\Re \setminus \{IQ, IV\}$
$ID\ (= IX^{-1})$	$\Re \setminus \{IQ, IS, IR, IJ, IL, IL', IL'', ID, ID', ID''\}$	$\Re \setminus \{IQ, IX\}$
$ID'\ (= IU^{-1})$	$\Re \setminus \{IQ, IS, IR, IJ, IL, IL', IL'', ID, ID', ID''\}$	$\Re \setminus \{IQ, IU\}$
$IE\ (= IW^{-1})$	$\Re \setminus \{IQ, IS, IR, IJ, IL, IL', IL'', ID, ID', ID'', IE\}$	$\Re \setminus \{IQ, IW\}$
$IE'\ (= IT^{-1})$	$\Re \setminus \{IQ, IS, IR, IJ, IL, IL', IL'', ID, ID', ID'', IE'\}$	$\Re \setminus \{IQ, IT\}$
$IF\ (= IS^{-1})$	$\Re \setminus \{IQ, IS, IR, IJ, IL, IL', IL'', ID, ID', ID'', IF\}$	$\Re \setminus \{IQ, IS\}$
$IG\ (= IG^{-1})$	$\Re \setminus \{IQ, IR, IJ, IV, IK, IG\}$	$\Re \setminus \{IQ, IR, IJ, IV, IK, IG\}$
$IH\ (= IK^{-1})$	$\Re \setminus \{IQ, IR, IJ, IV, IK, IG, IH\}$	$\Re \setminus \{IQ, IR, IJ, IK\}$
$II\ (= IJ^{-1})$	$\Re \setminus \{IQ, IR, IJ, IV, IK, IG, II\}$	$\Re \setminus \{IQ, IR, IJ\}$
$IL\ (= IO^{-1})$	$\Re \setminus \{IQ, IR, IJ, IL\}$	$\Re \setminus \{IQ, IR, IJ, IO\}$
$IL'\ (= IP^{-1})$	$\Re \setminus \{IQ, IR, IJ, IL'\}$	$\Re \setminus \{IQ, IR, IJ, IP\}$
$IM\ (= IM^{-1})$	$\Re \setminus \{IQ, IR, IJ, IM\}$	$\Re \setminus \{IQ, IR, IJ, IM\}$
$IN\ (= IM'^{-1})$	$\Re \setminus \{IQ, IR, IJ, IN\}$	$\Re \setminus \{IQ, IR, IJ, IM'\}$
$IN'\ (= IN'^{-1})$	$\Re \setminus \{IQ, IR, IJ, IN'\}$	$\Re \setminus \{IQ, IR, IJ, IN'\}$
$ID''\ (= (IUX)^{-1})$	$\Re \setminus \{IQ, IS, IR, IJ, IL, IL', IL'', ID, ID', ID''\}$	$\Re \setminus \{IQ, IUX\}$
$IE''\ (= (ITW)^{-1})$	$\Re \setminus \{IQ, IS, IR, IJ, IL, IL', IL'', ID, ID', ID'', IE''\}$	$\Re \setminus \{IQ, ITW\}$
$IL''\ (= (IOP)^{-1})$	$\Re \setminus \{IQ, IR, IJ, IL''\}$	$\Re \setminus \{IQ, IR, IJ, IOP\}$
$IM''\ (= (IMN)^{-1})$	$\Re \setminus \{IQ, IR, IJ, IM''\}$	$\Re \setminus \{IQ, IR, IJ, IMN\}$
$IN''\ (= (IMN')^{-1})$	$\Re \setminus \{IQ, IR, IJ, IN''\}$	$\Re \setminus \{IQ, IR, IJ, IMN'\}$
$IMN''\ (= (IMN'')^{-1})$	$\Re \setminus \{IQ, IR, IJ, IMN''\}$	$\Re \setminus \{IQ, IR, IJ, IMN''\}$

The upper part of the table gives the function $\mathcal{H}$ on 29 relations assuming dense time. The lower part of the table gives function $\mathcal{H}$ for the 11 additional relations assuming nondense time.

theorem can be observed to be true from Lemma 1 and Table 5 by a case-by-case analysis.

Theorem 4. *For $R', R'' \in \Re$, if $R' \rightsquigarrow R''$, then $R'^{-1} \neq R''^{-1}$.*

Example. $IC \rightsquigarrow IB \Rightarrow IV(= IC^{-1}) \neq IR(= IB^{-1})$, which is true. Note that $R' \neq R''$ in the definition of relation $\rightsquigarrow$ is necessary; otherwise, $R' \rightsquigarrow R'$ will become true and, from Theorem 4, we get $R'^{-1} \neq R'^{-1}$, which leads to a contradiction.

Lemma 2. *If the relationship $R(X, Y)$ between intervals X and Y (at processes P_i and P_j, respectively) is contained in the set $\mathcal{H}(r_{i,j})$, then interval X can be removed from the queue Q_i.*

Proof. From the definition of $\mathcal{H}(r_{i,j})$, we get that $r_{i,j}(X, Y')$ cannot exist, where Y' is any successor interval of Y. Hence, interval X can never be a part of the solution and can be deleted from the queue. □

Lemma 3. *If the relationship between a pair of intervals X and Y (at processes P_i and P_j, respectively) is not equal to $r_{i,j}$, then either interval X or interval Y can be removed from the queue Q_i or Q_j, respectively.*

Proof. We use contradiction. Assume relation $R(X, Y)$ $(\neq r_{i,j}(X, Y))$ is true for intervals X and Y. From Lemma 2, the only time neither X nor Y will be deleted is when $R \notin \mathcal{H}(r_{i,j})$ and $R^{-1} \notin \mathcal{H}(r_{j,i})$. From Lemma 1, it can be inferred that $R \rightsquigarrow r_{i,j}$ and $R^{-1} \rightsquigarrow r_{j,i}$. As $r_{i,j}^{-1} = r_{j,i}$, we get $R \rightsquigarrow r_{i,j}$ and $R^{-1} \rightsquigarrow r_{i,j}^{-1}$. This is a contradiction as, by Theorem 4, $R \rightsquigarrow r_{i,j} \Rightarrow R^{-1} \neq r_{i,j}^{-1}$. Hence, $R \in \mathcal{H}(r_{i,j})$ or $R^{-1} \in \mathcal{H}(r_{j,i})$ or both and, thus, at least one of the intervals can be deleted. □

Observe from Table 5 that it is possible that both intervals being compared can be deleted.

Example. If $r_{i,j} = IB$ is to be detected and it is determined that $R(X_i, Y_j)$ holds, where

$$R(X_i, Y_j) \in \{IF, II, IP, IO, IU, IX, IUX, IOP\} \subset \mathcal{H}(r_{i,j}),$$

then X_i should be deleted. Also, we have that

$$R^{-1}(Y_j, X_i) \in \{IF^{-1}, IP^{-1}, IO^{-1}, IU^{-1}, IX^{-1}, IUX^{-1}, IOP^{-1}\} \subset \mathcal{H}(r_{j,i})$$

and, hence, Y_j should also be deleted. This can also be seen from Fig. 1.

Theorem 5. *Algorithm Fine_Rel run by P_0 in Fig. 6 solves Problem Fine_Rel.*

Proof sketch. Lemma 2, which allows queues to be pruned correctly, is implemented in the algorithm at P_0. The code shown is executed atomically when an interval arrives at P_0. The orthogonal interaction $R(X, Y)$ between a pair of intervals X and Y being considered is determined in line 12. This determination is done by evaluating R1 to R4 using the tests in Table 2 and by evaluating S1 and S2 using the tests in Fig. 5 that implement their definitions in Table 2. Using this information, the bit-vector in Table 3 identifies the orthogonal interaction type between X and Y. The algorithm deletes interval X as soon as $R(X, Y) \in \mathcal{H}(r_{i,j})$ (lines 13-17). Similarly, Y is deleted if $R(Y, X) \in \mathcal{H}(r_{j,i})$ (lines 15-17). Thus, an interval gets deleted only if it cannot be part of the solution. Also, clearly, each

For $S2(X, Y)$

1. // Eliminate from Log of interval Y (on P_j), all receives of messages
   //which were sent by P_i before the start of interval X (on P_i).
   (1a) **for** each $event\_interval \in Log_j.p\_log[i].event\_interval\_queue$
   (1b) **if** $(event\_interval.interval\_id < Log_i.start[i])$ **then** delete $event\_interval$.
2. // Select from the pruned Log, the earliest message sent from interval X to Y.
   (2a) $temp = \infty$
   (2b) **if** $(Log_j.start[i] \geq Log_i.start[i])$ **then** $temp = Log_j.start[j]$
   (2c) **else**
   (2d) **for** each $event\_interval \in Log_j.p\_log[i].event\_interval\_queue$
   (2e) $temp = min(temp, event\_interval.local\_event)$.
3. **if** $(Log_i.end[j] \geq temp)$ **then** $S2(X, Y)$ is true.

For $S1(Y, X)$

1. Same as step 1 of the algorithm to determine $S2(X, Y)$.
2. Same as step 2 of the algorithm to determine $S2(X, Y)$.
3. **if** $(Log_i.end[j] < temp)$ and $(temp > Log_j.start[j])$ **then** $S1(Y, X)$ is true.

Fig. 5. Tests for coupling relations $S1(X, Y)$ and $S2(Y, X)$ at P_0. The tests implement the definitions given in Table 2.

queue of Log: $Q_1, Q_2, \ldots Q_n = \perp$
set of int: $updatedQueues$, $newUpdatedQueues = \{\}$

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** ($updatedQueues$ is not empty)
(5) $newUpdatedQueues = \{\}$
(6) **for** each $i \in updatedQueues$
(7) **if** (Q_i is non-empty) **then**
(8) X = head of Q_i
(9) **for** $j = 1$ to n $(i \neq j)$
(10) **if** (Q_j is non-empty) **then**
(11) Y = head of Q_j
(12) Determine $R(X, Y)$, where $R \in \Re$, using the tests in Tab. 2, Fig. 5, Tab. 3
(13) **if** $(R(X, Y) \in \mathcal{H}(r_{i,j}))$ **then**
(14) $newUpdatedQueues = \{i\} \cup newUpdatedQueues$
(15) **if** $(R(Y, X) \in \mathcal{H}(r_{j,i}))$ **then**
(16) $newUpdatedQueues = \{j\} \cup newUpdatedQueues$
(17) Delete heads of all Q_k, where $k \in newUpdatedQueues$
(18) $updatedQueues = newUpdatedQueues$
(19) **if** (all queues are non-empty) **then**
(20) solution found. Heads of queues identify intervals that form the solution.

Fig. 6. Online algorithm $Fine\_Rel$ at the data fusion server P_0.

interval gets processed unless a solution is found using a predecessor interval from the same process. Lemma 3 gives the unique property that if $R(X, Y) \neq r_{i,j}$, then either interval X or interval Y is deleted. A consequence of this property is that if *every* queue is nonempty and their heads cannot be pruned, then a solution exists and the set of intervals at the head of each queue forms a solution.

The set $updatedQueues$ stores the indices of all the queues whose heads got updated. In each iteration of the **while** loop, the indices of all the queues whose heads satisfy Lemma 2 are stored in set $newUpdatedQueues$ (lines 13-16). In lines 17 and 18, the heads of all these queues are deleted and indices of the updated queues are stored in the set $updatedQueues$. Observe that only interval pairs which were not compared earlier are compared in subsequent iterations of the **while** loop. The loop runs until no more queues can be updated. If, at this stage, *all* the queues are nonempty, then a solution is found (follows from Lemma 3). If a solution is found, then, for the intervals X (at P_i) and Y (at P_j) stored at the heads of these queues, we have $R(X,Y) = r_{i,j}$. □

Theorem 6. *Algorithm $Fine\_Rel$ has the following complexities:*

- *The total space complexity at process P_0 is $O(min((4n-2)np, (6m_s+4m_r)n))$.*
- *The average time complexity at P_0 is $O((n-1)min((2m_s+2m_r), pn))$. This is equivalent to $O(n^2M)$, where M is maximum number of entries in a queue.*

Proof. For the central process P_0, the total space required is $O(min((4n-2)np, (6m_s+4m_r)n))$ because the total space overhead at P_0 is equal to the total message space complexity (see Theorem 3).

The time complexity is the product of the number of steps required to determine an orthogonal relationship from $\Re$ and the number of relationships determined.

Consider the first part of the product:

- The total number of interval pairs between any two processes P_i and P_j is p^2. To determine $R1(X,Y)$ to $R4(X,Y)$ and $R1(Y,X)$ to $R4(Y,X)$, as eight comparisons are needed for each interval pair, $8p^2$ comparisons are necessary for any pair of processes.
- To determine the number of comparisons required by $S1$ and $S2$, consider the maximum number of $Event\_Intervals$ stored in $Log_j.p\_log[i]$ that are sent over the execution lifetime to the central process as part of the Logs. This is the maximum number of $Event\_Intervals$ corresponding to P_i stored in Q_j over P_j's execution lifetime. An $Event\_Interval$ is added to $Log_j.p\_log[i]$ only when there is a change in the ith component of $Interval\ Clock$ at the receive of a message. As the ith component of $Interval\ Clock$ changes only when a new interval starts, the total number of times the ith component of $Interval\ Clock$ changes is at most equal to p, the maximum number of intervals occurring on the other process P_i. From Fig. 5, it can be observed that, for each $Event\_Interval$, there is one comparison. Thus, to determine the relationship between an interval on P_j and all other intervals on P_i, the number of comparisons is equal to p. As there are p intervals on P_j, a total of p^2 comparisons are required to determine $S1$ or $S2$. Hence, the total number of comparisons to determine $S1(X,Y)$, $S2(X,Y)$, $S1(Y,X)$, and $S2(Y,X)$ is $4p^2$.

This gives a total of $8p^2 + 4p^2 = O(p^2)$ comparisons to determine the relation between each pair of intervals on a pair of processes. As there are a total of p^2 intervals pairs between two processes, the average number of comparisons required to determine a relationship is $O(1)$.

To analyze the second part of the product, consider Fig. 6. For each interval considered from one of the queues in $updatedQueues$ (lines 6-12), the number of relations determined is $n-1$. Thus, the number of relations determined for each iteration of the **while** loop is $(n-1)|updatedQueues|$, where $|updatedQueues|$ denotes the number of entries in $updatedQueues$. The cumulative $\sum |updatedQueues|$ over all iterations of the **while** loop is less than the total number of intervals over all the queues. Thus, the total number of relations determined is less than $(n-1)min(2m_s+2m_r, pn)$, where $min(2m_s+2m_r, np)$ is the upper bound on the total number of intervals over all the queues. As the average time required to determine a relationship is $O(1)$, the average time complexity of the algorithm is equal to $O((n-1)min(2m_s+2m_r, pn))$.

The average time complexity is equivalently expressed using M, the maximum number of entries in a queue, as follows: The total number of intervals over all the queues is $O(nM)$. As the total number of relations determined is $(n-1)\sum |updatedQueues|$ over all the iterations of the **while** loop, this is equivalent to $(n-1).nM = O(n^2M)$. This is also the average time complexity as it takes $O(1)$ time on average to determine a relationship. □

Table 1 compares the complexities of $Fine\_Rel$ with those of earlier algorithms [15], [16] to detect $Possibly$ and $Definitely$. The earlier algorithms computed their time complexity at P_0 as $O(n^2M)$, not in terms of m_s, m_r, or p. They did not give the space complexity at P_0. As each control message in these earlier algorithms carries a fixed size $O(n)$ message overhead and a control message is sent to P_0 for every message send/receive event, we computed their total space complexity and average time complexity at P_0 as $O(nm_r)$. This enables a direct comparison with the our algorithm. Further, we have also computed our average time complexity, using M, as $O(n^2M)$. In our algorithm, note that $M \leq p$; $M = p$ if the message overhead optimization is not used. We do not express the total space at P_0 in terms of M because the queue entries are of variable size, with an average size of $(4n-2)$ integers.

5 ALGORITHMS $Fine\_Poss$ AND $Fine\_Def$

By leveraging Theorem 1 and the mapping of fine-grained modalities to $Possibly$ and $Definitely$ modalities, as given in Table 4, we address the problems of determining whether $Possibly(\phi)$ and $Definitely(\phi)$ hold. If either of these two coarse-grained modalities holds, we can also determine the exact fine-grained orthogonal relation/modality between each pair of processes, unlike any previous algorithm. Further, the time, space, and message complexities of the proposed online (centralized) detection algorithms (Algorithms $Fine\_Poss$ and $Fine\_Def$) to detect $Possibly$ and

(13) **if** $(R(X,Y) \in \bigcap_{r_{i,j} \in r^*_{i,j}} \mathcal{H}(r_{i,j}))$ **then**
(14) $\quad newUpdatedQueues = \{i\} \cup newUpdatedQueues$
(15) **if** $(R(Y,X) \in \bigcap_{r_{j,i} \in r^*_{j,i}} \mathcal{H}(r_{j,i}))$ **then**
(16) $\quad newUpdatedQueues = \{j\} \cup newUpdatedQueues$

Fig. 7. Algorithm $Fine\_Rel'$: Changes to Algorithm $Fine\_Rel$ are listed, assuming $r^*_{i,j}$ satisfies Property $\mathcal{CONVEXITY}$.

$Definitely$ in terms of the fine-grained modalities per pair of processes are the *same as* those of the earlier online (centralized) algorithms [15], [16] that can detect only whether the $Possibly$ and $Definitely$ modalities hold.

Recall that $\Re$ is a set of orthogonal relations and, hence, one and only one relation from $\Re$ must hold between any pair of intervals. Consider the case where, for each pair of processes (P_i, P_j), there is a set $r^*_{i,j} \subseteq \Re$ such that some relation in $r^*_{i,j}$ must hold. Now, consider the objective where we need to identify one interval per process such that, for each process pair (P_i, P_j), some relation in $r^*_{i,j}$ holds for that (P_i, P_j). We formalize this objective by generalizing the detection problem $Fine\_Rel$ to problem $Fine\_Rel'$, as follows:

Problem $Fine\_Rel'$ **Statement.** *Given a set of relations $r^*_{i,j} \subseteq \Re$ for each pair of processes P_i and P_j, determine online the intervals, if they exist, one from each process, such that any one of the relations in $r^*_{i,j}$ is satisfied (by the intervals) for each (P_i, P_j) pair. If a solution exists, identify the fine-grained interaction from $\Re$ for each pair of processes.*

To solve $Fine\_Rel'$, given an *arbitrary* $r^*_{i,j}$, a solution based on Algorithm $Fine\_Rel$ (Fig. 6) will not work because, in the crucial tests in lines 13-14, neither interval may be removable and yet none of the relations from $r^*_{i,j}$ might hold between the two intervals. This leads to deadlock! To see this further, let $r1, r2 \in r^*_{i,j}$ and let $R(X,Y)$ hold, where $R \notin r^*_{i,j}$. Now, let $R \in \mathcal{H}(r1)$, $R^{-1} \notin \mathcal{H}(r1^{-1})$, $R \notin \mathcal{H}(r2)$, $R^{-1} \in \mathcal{H}(r2^{-1})$. Interval X cannot be deleted because $R \notin \mathcal{H}(r2)$, implying that $r2(X,Y')$ may be true for a successor Y' of Y. Interval Y cannot be deleted because $R^{-1} \notin \mathcal{H}(r1^{-1})$, implying that $r1^{-1}(Y,X')$ may be true for a successor X' of X. Therefore, a solution based on Algorithm $Fine\_Rel$ will deadlock and a more elaborate (and presumably more expensive) solution will be needed.

Example. Let $r^*_{i,j} = \{IB, IG\}$ and let $R(X,Y) = IH$. Neither X nor Y can be deleted.

We now identify and define a special property, termed $\mathcal{CONVEXITY}$, on $r^*_{i,j}$ such that the deadlock is prevented. Informally, this property says that there is no relation R outside $r^*_{i,j}$ such that, for any $r1, r2 \in r^*_{i,j}$, $R \backsim r1$ and $R^{-1} \backsim r2^{-1}$. This property guarantees that, when intervals X and Y are compared for $r^*_{i,j}$ and $R(X,Y)$ holds, either X or Y or both get deleted and, hence, there is progress. The sets $r^*_{i,j}$, derived from Table 4, that need to be detected to solve Problems *Fine_Poss* and *Fine_Def* satisfy this property. We therefore observe that problems *Fine_Poss* and *Fine_Def* are special cases of Problem $Fine\_Rel'$ in which the property $\mathcal{CONVEXITY}$ on $r^*_{i,j}$ is necessarily satisfied. To solve Problems $Fine\_Poss$ and $Fine\_Def$, we then use the generalizations of Lemmas 2 and 3, as given in Lemmas 4 and 5, respectively, to first solve $Fine\_Rel'$.

Definition 4 (Convexity).

$$\mathcal{CONVEXITY} : \forall R \notin r^*_{i,j} : \Big(\forall r_{i,j} \in r^*_{i,j}, R \in \mathcal{H}(r_{i,j}) \bigvee \forall r_{j,i} \in r^*_{j,i}, R^{-1} \in \mathcal{H}(r_{j,i})\Big).$$

Lemma 4. *If the relationship $R(X,Y)$ between intervals X and Y (at processes P_i and P_j, respectively) is contained in the set $\bigcap_{r_{i,j} \in r^*_{i,j}} \mathcal{H}(r_{i,j})$, then interval X can be removed from the queue Q_i.*

Proof. From the definition of $\mathcal{H}(r_{i,j})$, we infer that no relation $r_{i,j}(X,Y')$, where $r_{i,j} \in r^*_{i,j}$ and Y' is any successor interval of Y on P_j, can be true. Hence, interval X can never be a part of the solution and can be deleted from the queue. □

Lemma 5. *If the relationship $R(X,Y)$ between a pair of intervals X and Y (at processes P_i and P_j, respectively) does not belong to the set $r^*_{i,j}$, where $r^*_{i,j}$ satisfies property $\mathcal{CONVEXITY}$, then either interval X or interval Y is removed from the queue.*

Proof. We use contradiction. Assume relation $R(X,Y)$ $(\notin r^*_{i,j}(X,Y))$ is true for intervals X and Y. From Lemma 4, the only time neither X nor Y will be deleted is when both $R \notin \bigcap_{r_{i,j} \in r^*_{i,j}} \mathcal{H}(r_{i,j})$ and $R^{-1} \notin \bigcap_{r_{j,i} \in r^*_{j,i}} \mathcal{H}(r_{j,i})$. However, as $r^*_{i,j}$ satisfies property $\mathcal{CONVEXITY}$, we have that $R \in \bigcap_{r_{i,j} \in r^*_{i,j}} \mathcal{H}(r_{i,j})$ or $R^{-1} \in \bigcap_{r_{j,i} \in r^*_{j,i}} \mathcal{H}(r_{j,i})$ must be true. Thus, at least one of the intervals can be deleted by an application of Lemma 4. □

The proof of the following theorem is similar to the proof of Theorem 5.

Theorem 7. *If the set $r^*_{i,j}$ satisfies property $\mathcal{CONVEXITY}$, then Problem $Fine\_Rel'$ is solved by replacing lines 13 and 15 in Algorithm $Fine\_Rel$ in Fig. 6 by lines 13 and 15 in Fig. 7.*

Proof. Analogous to the proof of Theorem 5. Use Lemmas 4 and 5 instead of Lemmas 2 and 3, respectively, and reason with $r^*_{i,j}$ instead of with $r_{i,j}$. □

Corollary 1. *The time, space, and message complexities of Algorithm $Fine\_Rel'$ are the same as those of Algorithm $Fine\_Rel$, which were stated in Theorem 6.*

Proof. The only changes to Algorithm $Fine\_Rel$ are in lines 13 and 15. In Algorithm $Fine\_Rel'$, instead of checking $R(X,Y)$ for membership in $\mathcal{H}(r_{i,j})$ in line 13, $R(X,Y)$ is checked for membership in $\bigcap_{r_{i,j} \in r^*_{i,j}} \mathcal{H}(r_{i,j})$. Both $\mathcal{H}(r_{i,j})$ and $\bigcap_{r_{i,j} \in r^*_{i,j}} \mathcal{H}(r_{i,j})$ are sets of size between 0

and $|\Re| = 40$. An analogous observation holds for the change on line 15. Hence, the time, space, and message complexities of $Fine\_Rel$ are unaffected in $Fine\_Rel'$. □

To detect $Possibly(\phi)$, $r^*_{i,j}$ is set to the union of the orthogonal interactions in the first two columns of Table 4. We can verify (by case-by-case enumeration) that $r^*_{i,j}$ does satisfy Property $\mathcal{CONVEXITY}$. Similarly, to detect $Definitely(\phi)$, $r^*_{i,j}$ is set to the union of the orthogonal interactions in the first column of Table 4. We can verify (by case-by-case enumeration) that $r^*_{i,j}$ does satisfy Property $\mathcal{CONVEXITY}$.

The following two theorems about using algorithm $Fine\_Rel'$ (Fig. 7) to solve Problems $Fine\_Poss$ and $Fine\_Def$ can be readily proven by using Theorem 1, the refinement mapping of Table 4, and Theorem 7. The two resulting algorithms are named $Fine\_Poss$ and $Fine\_Def$, respectively.

Theorem 8. *Algorithm $Fine\_Rel$ modified to Algorithm $Fine\_Rel'$ (Fig. 7) solves Problem $Fine\_Poss$ (about $Possibly(\phi)$) when $r^*_{i,j}$ is set to the union of the relations in the first and second columns of Table 4.*

Proof. From Theorem 1, $Possibly(\phi)$ is true if and only if $(\forall i \in N)(\forall j \in N) Possibly(\phi_i \wedge \phi_j)$. For any i and j, $Possibly(\phi_i \wedge \phi_j)$ is true if and only if $R(X_i, Y_j)$ is any of the temporal relations given in the first two columns of Table 4. When $r^*_{i,j}$ is set to the union of the relations in these two columns, we can verify (by case-by-case enumeration) that $r^*_{i,j}$ satisfies $\mathcal{CONVEXITY}$. As Algorithm $Fine\_Rel'$ is correct (by Theorem 7), when its $r^*_{i,j}$ is instantiated with the set above to get Algorithm $Fine\_Poss$, we have that $Fine\_Poss$ is also correct. □

Theorem 9. *Algorithm $Fine\_Rel$ modified to Algorithm $Fine\_Rel'$ (Fig. 7) solves Problem $Fine\_Def$ (about $Definitely(\phi)$) when $r^*_{i,j}$ is set to the union of the relations in the first column of Table 4.*

Proof. From Theorem 1, $Definitely(\phi)$ is true if and only if $(\forall i \in N)(\forall j \in N) Definitely(\phi_i \wedge \phi_j)$. For any i and j, $Definitely(\phi_i \wedge \phi_j)$ is true if and only if $R(X_i, Y_j)$ is any of the temporal relations given in the first column of Table 4. When $r^*_{i,j}$ is set to the relations in this column, we can verify (by case-by-case enumeration) that $r^*_{i,j}$ satisfies $\mathcal{CONVEXITY}$. As Algorithm $Fine\_Rel'$ is correct (by Theorem 7), when its $r^*_{i,j}$ is instantiated with the set above to get Algorithm $Fine\_Def$, we have that $Fine\_Def$ is also correct. □

In Algorithm $Fine\_Rel'$, when $r^*_{i,j}$ is set to the values as specified in Theorems 8 and 9 to detect *Possibly* and *Definitely*, respectively, the set $\bigcap_{r_{i,j} \in r^*_{i,j}} \mathcal{H}(r_{i,j})$ used in line 13 of the algorithm becomes $\{IA\}$ and $\{IA, IB, IC, IG, IH, II, IK, IV, IM'', IMN'', IMN\}$, respectively. An identical change occurs to the set $\bigcap_{r_{j,i} \in r^*_{j,i}} \mathcal{H}(r_{j,i})$ on line 15.

Corollary 2. *The time, space, and message complexities of Algorithms $Fine\_Poss$ and $Fine\_Def$ are the same as those of Algorithm $Fine\_Rel$ (stated in Theorem 6) and of Algorithm $Fine\_Rel'$ (stated in Corollary 1).*

Proof. Follows from Corollary 1 and the fact that $r^*_{i,j}$ for $Fine\_Poss$ and $Fine\_Def$ satisfy $\mathcal{CONVEXITY}$ and are instantiations of $r^*_{i,j}$ in $Fine\_Rel'$. □

6 Algorithm $All\_Pairs\_Fine\_Rel$

Using the tests in Table 2 and Fig. 5, we can determine the 12-bit pattern of Table 3 and, hence, the interaction type between a pair of intervals. To determine the relations between all pairs of intervals from different processes, the total number of intervals stored at P_0 across all the queues Q_1 to Q_n in the worst case is pn. The number of runs of the tests required to determine the relation between all pairs of intervals is equal to the total number of pairs of intervals, which is $C^n_2.p^2$. To reduce the memory overhead and the number of tests, the following queue pruning scheme is used.

Lemma 6. *If the relation R between an interval X and interval Y is contained in the set $\mathcal{S} = \{r \in \Re \mid r(X,Y) \Rightarrow R2(X,Y) = 1$ (defined in Table 3) $\} = \{IA, IB, II, IX, IO, IP, IF, IU, IUX, IOP\}$, then X and the successors of interval Y being queued on Q_j are related by IA and no further comparison is needed between them.*

Proof. As the control messages from any process are received by P_0 in FIFO order, the interval Y' that gets queued after Y in Q_j is a successor of Y. This, coupled with the fact that $R(X,Y) \in \mathcal{S}$, implies $V^+_i(X)$ is less than $V^+_j(Y)$, which in turn implies $V^-_j(Y')$ is greater than $V^+_i(X)$. Thus, the only relation possible between X and Y' is IA. □

Using Lemma 6, we can reduce the number of comparisons.

Lemma 7. *If Lemma 6 is satisfied for an interval X with respect to at least one interval on each of the other queues, then interval X can be removed from its queue.*

Proof. As Lemma 6 is satisfied for interval X with respect to at least one interval Y_j on each other queue Q_j, we conclude that $R(X, Y'_j) = IA$ between X and any subsequent interval Y'_j after Y_j on Q_j. Hence, we can remove X from its queue. □

The above lemma is used to decrease the memory storage overhead for the central process P_0. Both of the above lemmas are used in the predicate detection algorithm at P_0. Fig. 8 gives an efficient algorithm to detect the relationship between each pair of intervals.

Theorem 10. *The algorithm run by P_0 in Fig. 8 solves Problem $All\_Pairs\_Fine\_Rel$.*

Proof sketch. An interval X received at P_0 is enqueued in Q_i and compared with intervals on all the other queues. The algorithm stops comparison of interval X with successors of interval Y as soon as X and Y satisfy Lemma 6. The interval X is pruned from the queue after it satisfies Lemma 7, i.e., it satisfies Lemma 6 with respect to all the processes. This is determined by keeping an n-bit array B for each of the intervals stored in the queues at process P_0. For interval X on queue Q_i, the n-bit array is represented by B^X_i. If $B^X_i[j]$ is set to 1 when comparing intervals X and Y (lines 8-9), then no

On receiving interval X from process P_i at P_0
(1) Enqueue the interval X onto queue Q_i
(2) $B_i^X[1..n] = 0$ //initialize boolean array of size n for this interval X_i
(3) **for** $j = 1$ to n
(4) **if** (Q_j is non-empty) **then**
(5) **for** each interval $Y \in Q_j$ starting from the head
(6) **if** ($B_j^Y[i] \neq 1$) **then** //Check if Lemma 6 applies
(7) Determine relation $R(X, Y)$, where $R \in \Re$, using tests in Tab. 2, Fig. 5, Tab. 3
(8) **if** ($R(X, Y) \in \mathcal{H}_{\Re \setminus \{IA\}}$) **then** $B_i^X[j] = 1$ // apply Lemma 6
(9) **if** ($R(Y, X) \in \mathcal{H}_{\Re \setminus \{IA\}}$) **then** $B_j^Y[i] = 1$ // apply Lemma 6
(10) **if** (all bits of $B_i^X = 1$) **then** dequeue X // apply Lemma 7
(11) **if** (all bits of $B_j^Y = 1$) **then** dequeue Y. // apply Lemma 7

Fig. 8. Algorithm $All\_Pairs\_Fine\_Rel$: to determine the relations between each pair of intervals.

comparison is needed between X and any successor of Y because the relationship between X and successors of Y will be IA. Thus, the relation between all interval pairs is explicitly determined except when it is known to be IA or IQ. □

Theorem 11. *Algorithm $All\_Pairs\_Fine\_Rel$ has the following complexities:*

- *The total message space complexity is $O(n^2p)$.*
- *The total space complexity at process P_0 is $O(n^2p)$.*
- *The average time complexity at P_0 is $O(n^2p^2)$.*

Proof. The total *message space* complexity can be shown as in Theorem 3 to be $O(n^2p)$. The message optimization used there is not applicable as all intervals need to be considered in *All_Pairs_Fine_Rel*.

The *space complexity* at P_0 is the space occupied by all the queues. This includes the total message space complexity and also the $O(n)$ space for each of the np instances of the B array. The space complexity at P_0 is thus $O(n^2p)$.

The *time complexity* is the product of the number of steps required to determine a relationship from $\Re$ and the number of relationships determined.

- It was shown in the proof of Theorem 6 that 1) a total of $O(p^2)$ comparisons are needed to determine a relationship between each pair of intervals on a given pair of processes and 2) it takes average time $O(1)$ to determine a relationship.
- There are C_2^n or $O(n^2)$ pairs of processes.

Hence, the average time complexity of the algorithm is $O(n^2p^2)$. The actual time complexity is expected to be much less as the number of comparisons may be greatly reduced by using Lemmas 6 and 7. □

7 Operation in Mobile Systems

The complexity of the algorithms to solve $Fine\_Rel$, $Fine\_Poss$, and $Fine\_Def$ is shown in Table 1. It can be seen that these algorithms have low overhead. All the measures at P_0 are either linear or quadratic in n, the size of the mobile ad hoc network. This indicates high scalability of the algorithms. Most importantly, the size of the space requirement at each process P_i ($i \in [1, n]$), which is both space and energy constrained, is linear in n. This makes the algorithm easily implementable and scalable.

Recall that mobility *is* implicitly permitted in the model (Section 2.1). We now show how to implement the algorithms in mobile ad hoc networks. The algorithms require each process P_i ($1 \leq i \leq n$) to eventually and asynchronously (via any routes) send its event stream to a central data fusion server process P_0 via a FIFO channel, which can be achieved in mobile ad hoc networks. We use hierarchical clustering to manage the network. Further scalability can be achieved by using "hierarchical" timestamps based on logical clocks. Several efficient implementations of vector clocks, such as the difference method [35] and others surveyed in [22], can also be used in conjunction.

In the hierarchical organization of the clusters, the data fusion server P_0 is the root and a logical tree is superimposed such that the mobile processes are the leaf nodes. Communication paths from the leaves to the root can be provided by the hierarchical mobile routing algorithms [2], [3], [4], [26], [30], [32]. The FIFO channels between each P_i ($i \neq 0$) and P_0 can also be implemented using sequence numbering with retransmissions at the application layer, rather than in the MAC layer. For a $k+1$ level hierarchy, each cluster at each level except the highest level can be organized to have approximately α nodes, where $n = \alpha^k$. The above algorithms [2], [3], [4], [26], [30], [32] can be used to maintain the hierarchical cluster organization despite mobility and failures. For example, for greater fault tolerance (to node failures), there can be some overlap among the clusters [3]. In particular, the following can be ensured using these algorithms:

- Despite mobility of the processes, the hierarchical tree structure is maintained. Thus, some clusterhead is always identified with each node, using handoff.
- Despite mobility, occasional lost connectivity, and node deaths, a FIFO logical channel is maintained between each P_i ($1 \leq i \leq n$) and P_0.
- If a node dies, the algorithm at P_0 is not blocked. If a node in the hierarchy dies, the subtree rooted at that node gets reconnected.

- The algorithm works seamlessly and correctly with backup data fusion server(s), in case of server failure. Event stream routing transparently flips over to the backup server while maintaining the FIFO property from each node to the server.

The algorithms proposed in this paper used online, centralized fusion of the data collected from distributed processes. Although completely distributed and symmetrical algorithms performing distributed data fusion can be designed, they are not inherently suitable for mobile ad hoc networks.

1. The full symmetry may replicate the data fusion work at all the mobile nodes.
2. Decision-making often requires extra "meta-information" after the fusion, and such meta-information is best kept at some server P_0 (with a few backup replicas for fault tolerance). Mobile nodes may not be capable of decision making and a known logical location for decision making is preferred.
3. Fully symmetrical algorithms tend to place higher overhead on all the participants, in terms of some/all of space, time, and message complexity.
4. A distributed algorithm requires communication between *any* pair of processes, thus requiring a more elaborate routing scheme (any-to-any) and not just between the leaves and the root. (Multicasting as in [10] may be used for the distributed algorithm, but multicast overheads would be incurred.)

8 Conclusions and Discussion

This paper proposed online algorithms for detecting conjunctive predicates and behavioral patterns that can be specified using a *rich palette of fine-grained temporal modalities*. Algorithm $Fine\_Rel$ solves a fundamental problem of identifying a global state satisfying fine-grained interaction modalities between each pair of processes. Algorithms $Fine\_Poss$ and $Fine\_Def$ not only detect $Possibly(\phi)$ and $Definitely(\phi)$, respectively, *but also (unlike previous algorithms)* return the pairwise fine-grained relations which exist between all the intervals in the solution set. The space, message and computational complexities of the previous works for conjunctive predicate detection, which can detect only $Possibly(\phi)$ [15] or only $Definitely(\phi)$ [16], are compared with our algorithms in Table 1. All the complexity measures for algorithms $Fine\_Poss$ and $Fine\_Def$ are the same as those for the algorithms [15] and [16], respectively. Thus, with the same overhead, algorithms $Fine\_Poss$ and $Fine\_Def$ do the *extra work* of finding the fine-grained relations which exist between the intervals contained in the solution set for $Possibly$ and $Definitely$. The output of *All_Pairs_Fine_Rel* can be used to mine patterns in the execution.

The space, communication, and time overheads of the algorithms are low, making them deployable in resource-constrained mobile ad hoc networks. The algorithms are highly scalable in that the space requirement at each mobile node is of the order of the size of the system. We showed how to run the algorithms in a mobile ad hoc network.

Two distributed algorithms have been proposed for $Fine\_Rel$. The first algorithm [5], [6] uses $O(n \min(np, 4mn))$ messages of size $O(n)$ each (here, m is the maximum number of messages sent by any process). The second algorithm [6], [17] uses $O(\min(np, 4mn))$ messages of size $O(n^2)$ each. For both the algorithms, the total space complexity across all the processes is $\min(4n^2p - 2np, 10n^2m)$, the worst-case space overhead per process is $\min(4np - 2p, 4mn^2 + 2mn - 2m)$, and the time complexity at a process is $O(\min(np, 4mn))$. Analogous to the approach in this paper, the distributed algorithms can be used to design distributed versions of $Fine\_Poss$ and $Fine\_Def$.

Alternate definitions [17] of $\sim$ and $\mathcal{H}$ that exclude the condition $R \neq r_{i,j}$ lead to a simpler structure of the allows relation. The formalism will need to change accordingly. For example, the condition on lines 13 and 15 in Fig. 6 would also require $R(X, Y) \neq r_{i,j}$ and $R(Y, X) \neq r_{j,i}$, respectively, to be true. As another example, the condition on lines 13 and 15 in Fig. 7 would also require $R(X, Y) \notin r^*_{i,j}$ and $R(Y, X) \notin r^*_{j,i}$, respectively, to be true.

A discussion of how the presented algorithms and approach can be used to detect nonconjunctive predicates, i.e., general relational predicates, is given in [21]. The key idea is to compose and redefine "intervals" at the data fusion server using the execution's state lattice.

The proposed algorithms assume that the processes can timestamp (using vector clocks) the events *within* the executions of the processes. Recent advances in cost-effective technology have led to a spurt in designing sensor networks and actuator networks [1], [31], [37], [38] to observe real-world physical phenomena. Such networks typically use synchronized physical clocks; [36] surveys physical clock synchronization schemes for wireless sensor networks. Events in the environment are *outside* of the processes. Causality-based properties could be detected if the sensors can 1) detect communication within the physical environment being sensed (*outside* the network) and 2) assign logical vector timestamps to the communication events and events in the environment at which the monitored variables change. Thus, if it were possible to manifest causal relations in the events and communication within the environment, each sensed location can be assigned a "time-line" be modeled as another "process," and be included in the causality analysis. It does not seem that this can be done.

Recent results on detecting predicates that are specified using physically synchronized clocks are presented in [23].

Acknowledgments

An earlier version of some portions of this paper appeared in the *Proceedings of the 2003 ASIAN Computing Science Conference* [8].

References

[1] I. Akyildiz, W. Su, Y. Sankarasubramanian, and E. Cayirci, "Wireless Sensor Networks: A Survey," *Computer Networks*, vol. 38, no. 4, pp. 393-422, 2002.

[2] A. Amis, R. Prakash, T. Vuong, and D. Huynh, "Max-Min D-Cluster Formation in Wireless Ad-Hoc Networks," *Proc. IEEE Infocom,* pp. 32-41, 2000.

[3] S. Banerjee and S. Khuller, "A Clustering Scheme for Hierarchical Control in Multi-Hop Wireless Networks," *Proc. IEEE Infocom,* pp. 1028-1037, Apr. 2001.

[4] D. Baker and A. Ephremedis, "The Architectural Organization of a Mobile Radio Network via a Distributed Algorithm," *IEEE Trans. Comm.,* vol. 29, pp. 1694-1701, Nov. 1981.

[5] P. Chandra and A.D. Kshemkalyani, "Detection of Orthogonal Interval Relations," *Proc. Ninth Int'l Conf. High-Performance Computing,* pp. 323-333, 2002.

[6] P. Chandra and A.D. Kshemkalyani, "Distributed Detection of Temporal Interval Interactions," Technical Report UIC-ECE-02-07, Univ. of Illinois at Chicago, Apr. 2002.

[7] P. Chandra and A.D. Kshemkalyani, "Distributed Algorithm to Detect Strong Conjunctive Predicates," *Information Processing Letters,* vol. 87, no. 5, pp. 243-249, Sept. 2003.

[8] P. Chandra and A.D. Kshemkalyani, "Global Predicate Detection under Fine-Grained Modalities," *Proc. ASIAN Computing Science Conf. 2003,* pp. 91-109, Dec. 2003.

[9] K.M. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems," *ACM Trans. Computer Systems,* vol. 3, no. 1, pp. 63-75, 1985.

[10] C.-C. Chiang, M. Gerla, and L. Zhang, "Tree Multicast Strategies in Mobile, Multihop Wireless Networks," *Mobile Networks and Applications (MONET),* vol. 3, pp. 193-207, 1999.

[11] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems Concepts and Design,* third ed. Addison-Wesley, 2001.

[12] R. Cooper and K. Marzullo, "Consistent Detection of Global Predicates," *Proc. ACM/ONR Workshop Parallel and Distributed Debugging,* pp. 163-173, May 1991.

[13] J. Elson and K. Romer, "Wireless Sensor Networks: A New Regime for Time Synchronization," *Proc. First Workshop Hot Topics in Networks (HotNets-I),* Oct. 2002.

[14] C.J. Fidge, "Timestamps in Message-Passing Systems that Preserve Partial Ordering," *Australian Computer Science Comm.,* vol. 10, no. 1, pp. 56-66, Feb. 1988.

[15] V.K. Garg and B. Waldecker, "Detection of Weak Unstable Predicates in Distributed Programs," *IEEE Trans. Parallel and Distributed Systems,* vol. 5, no. 3, pp. 299-307, Mar. 1994.

[16] V.K. Garg and B. Waldecker, "Detection of Strong Unstable Predicates in Distributed Programs," *IEEE Trans. Parallel and Distributed Systems,* vol. 7, no. 12, pp. 1323-1333, Dec. 1996.

[17] P. Chandra and A.D. Kshemkalyani, "Global State Detection Based on Peer-to-Peer Interactions," *Proc. IFIP Int'l Conf. Embedded and Ubiquitous Computing,* Dec. 2005.

[18] M. Hurfin, M. Mizuno, M. Raynal, and M. Singhal, "Efficient Distributed Detection of Conjunctions of Local Predicates," *IEEE Trans. Software Eng.,* vol. 24, no. 8, pp. 664-677, Aug. 1998.

[19] A.D. Kshemkalyani, "Temporal Interactions of Intervals in Distributed Systems," *J. Computer and System Sciences,* vol. 52, no. 2, pp. 287-298, Apr. 1996.

[20] A.D. Kshemkalyani, "A Fine-Grained Modality Classification for Global Predicates," *IEEE Trans. Parallel and Distributed Systems,* vol. 14, no. 8, pp. 807-816, Aug. 2003.

[21] A.D. Kshemkalyani, "A Note on Fine-Grained Modalities for Nonconjunctive Predicates," *Proc. Fifth Int'l Workshop Distributed Computing,* pp. 11-25, Dec. 2003.

[22] A.D. Kshemkalyani, "The Power of Logical Clock Abstractions," *Distributed Computing,* vol. 17, no. 2, pp. 131-150, Aug. 2004.

[23] A.D. Kshemkalyani, "Predicate Detection Using Event Streams in Ubiquitous Environments," *Proc. Int'l Symp. Network-Centric Ubiquitous Systems,* Dec. 2005.

[24] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Comm. ACM,* vol. 21, no. 7, pp. 558-565, July 1978.

[25] L. Lamport, "On Interprocess Communication, Part I: Basic Formalism; Part II: Algorithms," *Distributed Computing,* vol. 1, pp. 77-85, vol. 1, pp. 86-101, 1986.

[26] C.R. Lin and M. Gerla, "Adaptive Clustering for Mobile Wireless Networks," *IEEE J. Selected Areas in Comm.,* vol. 15, no. 7, pp. 1265-1275, Sept. 1997.

[27] T. Logsdon, *The Navstar Global Positioning System.* New York: Van Nostrand/Reinhold, 1992.

[28] F. Mattern, "Virtual Time and Global States of Distributed Systems," *Parallel and Distributed Algorithms,* pp. 215-226, North-Holland, 1989.

[29] D. Mills, "Internet Time Synchronization: The Network Time Protocol," *IEEE Trans. Comm.,* vol. 39, no. 10, pp. 1482-1493, Oct. 1991.

[30] G. Pei, M. Gerla, X. Hong, and C. Chiang, "A Wireless Hierarchical Routing Protocol with Group Mobility," *IEEE Wireless Comm. and Networking Conference (WCNC),* pp. 1538-1542, 1999.

[31] G. Pottie and W. Kaiser, "Wireless Integrated Network Sensors," *Comm. ACM,* vol. 43, no. 5, pp. 51-58, May 2000.

[32] R. Ramanathan and M. Steenstrup, "Hierarchically Organized, Multihop Wireless Mobile Wireless Networks for Quality-Of-Service Support," *Mobile Networks and Applications (MONET),* vol. 3, no. 1, pp. 101-110, June 1998.

[33] M. Raynal and M. Singhal, "Logical Time: Capturing Causality in Distributed Systems," *Computer,* vol. 29, no. 2, pp. 49-56, Feb. 1996.

[34] R. Schwarz and F. Mattern, "Detecting Causal Relationships in Distributed Computations: In Search of the Holy Grail," *Distributed Computing,* vol. 7, pp. 149-174, 1994.

[35] M. Singhal and A.D. Kshemkalyani, "Efficient Implementation of Vector Clocks," *Information Processing Letters,* vol. 43, pp. 47-52, Aug. 1992.

[36] B. Sundararaman, U. Buy, and A.D. Kshemkalyani, "Clock Synchronization for Wireless Sensor Networks: A Survey," *Ad-Hoc Networks,* vol. 3, no. 3, pp. 281-323, May 2005.

[37] S. Tilak, N. Abu-Ghazaleh, and W. Heinzelman, "A Taxonomy of Wireless Micro-Sensor Models," *ACM Mobile Computing & Comm. Rev.,* vol. 6, no. 2, Apr. 2002.

[38] X. Yu, K. Niyogi, S. Mehrotra, and N. Venkatasubramanian, "Adaptive Middleware for Distributed Sensor Environments," *IEEE Distributed Systems Online,* vol. 4, May 2003.

Punit Chandra received the BTech degree in electrical engineering from the Indian Institute of Technology, Kanpur, in 1999 and the MS degree in computer science from the University of Illinois at Chicago in 2001. Currently, he is in the doctoral program at the University of Illinois at Chicago, and works for Siemens. His research interests are in distributed computing, computer networks, and operating systems.

Ajay D. Kshemkalyani received the PhD degree in computer and information science from The Ohio State University in 1991 and the BTech degree in computer science and engineering from the Indian Institute of Technology, Bombay, in 1987. His research interests are in computer networks, distributed computing, algorithms, and concurrent systems. He has been an associate professor at the University of Illinois at Chicago since 2000, prior to which he spent several years at IBM Research Triangle Park working on various aspects of computer networks. He is a member of the ACM and a senior member of the IEEE. In 1999, he received the US National Science Foundation's CAREER Award. He is currently on the Editorial Board of the Elsevier journal *Computer Networks*.