2013 IEEE 27th International Symposium on Parallel & Distributed Processing Workshops and PhD Forum

A Fault-Tolerant Strong Conjunctive Predicate Detection Algorithm for Large-Scale Networks

Min Shen and Ajay D. Kshemkalyani
Dept. of Computer Science, University of Illinois at Chicago,
Chicago, IL 60607-7053, USA
Email: {mshen6, ajay}@uic.edu

***Abstract*—In large-scale networks where a continuously ongoing monitoring program is needed, using traditional predicate detection algorithms might cause the system to have a single point of failure. This paper presents an on-line distributed algorithm that detects strong conjunctive predicates and we show that it is resilient to node failures. Our algorithm assumes a pre-constructed spanning tree in the system, and detects all satisfactions of the predicate in a hierarchical manner. Our algorithm is able to detect predicates at each level in the hierarchy, thus becoming resilient to node failures because of the capability to detect a partial predicate of the global predicate. This hierarchical detection manner also provides a finer-grained monitoring in those large-scale networks where grouping is established and the monitoring happens at the group level. Furthermore, comparing with other detection algorithms, our algorithm incurs a low space/time cost, which is distributed across all the nodes in the network, and a low message complexity. This makes our algorithm applicable in a resource-constraint network.**

***Keywords*-distributed system; predicate detection; fault-tolerant; large-scale network**

I. INTRODUCTION

Detecting predicates over a distributed execution is important for various purposes such as monitoring, synchronization, coordination, and debugging. In recent years, predicate detection has found applications in large-scale networks such as WSNs [1] and modular robotics [2], [3], where individual nodes have only limited computation resources and are subject to node failures. With these properties, new solutions that conserve the limited resources and take into consideration the potential failures of the nodes are needed.

There are many predicate types and detection algorithms studied in the literature for traditional distributed systems (see [4] for a survey). One way to categorize a predicate is based on the function on the variables involved in the predicate [5]:

1) A *relational predicate* is a predicate that is expressed as an arbitrary relation on the variables in the system. Let x_i and y_j be local variables at process P_i and P_j, respectively. $\Phi =$ "$avg(x_i, y_j) = 35$" is a relational predicate.
2) A *conjunctive predicate* is a predicate that can be expressed as the conjunction of local predicates. $\Psi =$ "$x_i > 20 \wedge y_j < 45$" is a conjunctive predicate.

Due to the asynchrony in message transmissions and in local executions, different executions of the same distributed program can generate different sequences of global states. Therefore, whether a predicate gets satisfied within all consistent observations of an execution or within some consistent observation of an execution, can be different. Thus, two modalities under which a predicate Φ can hold [6] have been defined.

1) *Possibly(Φ)*: There exists a consistent observation of the execution such that Φ holds in a global state of the observation.
2) *Definitely(Φ)*: For every consistent observation of the execution, there exists a global state of it in which Φ holds. This type of predicates is also called strong predicates in [7].

Algorithms to detect both *Possibly(Φ)* and *Definitely(Φ)* for a conjunctive or relational predicate are given in [6]. However, it has been shown that detecting a relational predicate is an NP-complete problem. Due to the exponential complexity of detecting relational predicates, most work on predicate detection is focused on conjunctive ones.

In [7], [8], Garg and Waldecker gave centralized algorithms to detect *Definitely (Φ)* and *Possibly(Φ)*, respectively. In [7], they presented an interval-based approach to detect *Definitely(Φ)*. In [4], an algorithm that adopts a unified approach to detect both *Possibly(Φ)* and *Definitely(Φ)* based on intervals was given. For a network of n processes and an execution in which the local predicate becomes true at most p times at a process, the detection algorithm in [4] has a space and time complexity of $O(pn^2)$. It also generates $O(pn)$ messages, each of size $O(n)$.

The above centralized algorithms has the drawback that all the computation occurs at a single process. This uneven distribution of time and space complexity makes such algorithms undesirable in systems where individual processes have limited resources. Several distributed algorithms were thus proposed. Garg and Chase [9] and Hurfin et al. [10] presented distributed algorithms to detect *Possibly(Φ)*. Both algorithms have space, time and message complexities being $O(mn^2)$, where m is the maximum number of messages sent by any process. Chandra and Kshemkalyani [11] gave

978-0-7695-4979-8/13 $26.00 © 2013 IEEE
DOI 10.1109/IPDPSW.2013.156

IEEE computer society

Authorized licensed use limited to: University of Illinois at Chicago Library. Downloaded on June 22,2020 at 22:42:12 UTC from IEEE Xplore. Restrictions apply.

a distributed algorithm for detecting *Definitely(Φ)*. Its space and time complexities are $O(\min(pn, mn^2))$ and its message complexity is $O(\min(pn^2, mn^2))$.

When detecting predicates in continuous monitoring programs, usually the application requires the monitoring program to raise an alarm *each time* the predicate occurs. In such cases, none of these algorithms [7], [8], [9], [10], [11] are applicable. As shown in [12], these algorithms can detect predicates only once and will hang after the initial detection. They cannot detect multiple occurrences because detecting subsequent occurrences is not simply rerunning those one-time detection algorithms, but requires elaborate processing to ensure safety and liveness. In [12], a centralized repeated detection algorithm which can detect all occurrences of *Definitely(Φ)* in $O(pn^3)$ time is given. However, all the time/space costs incurred by this algorithm happen at the sink. In networks where individual nodes only have limited computation resources, the centralized algorithm is not desirable. Furthermore, this algorithm, as also the other centralized detection algorithms [4], [7], [8], will fail when one node fails. Even the distributed detection algorithms [9], [10], [11] have the same problem because there will be no progress after any one node failure.

In this paper, we present a decentralized algorithm that detects *Definitely(Φ)* within a large-scale network. This algorithm assumes a pre-constructed spanning tree in the network and detects the predicate in a hierarchical manner. By establishing a hierarchy in the network, our algorithm divides the task of detecting a predicate among different levels in the hierarchy. Each node detects the predicate within the subtree rooted at itself. If one node fails, the detection of the predicate in the remaining processes could be easily resumed because the hierarchical detection manner gives our algorithm the ability to detect a partial predicate of the global predicate. In addition, our algorithm detects the predicates in a repeated manner [12]. In long-running applications where continuous monitoring is required, repeated detection is essential because manual intervention after one detection of predicate satisfaction to reset the detection algorithm is not practical or even possible. Furthermore, the hierarchical structure of our algorithm can also provide a finer-grained monitoring in those large-scale networks where grouping is established and the monitoring is needed at the group level.

For a spanning tree of height h and degree d, our hierarchical algorithm has a global time complexity of $O(d^2pn^2)$ and a global space complexity of $O(pn^2)$, distributed across all nodes in the network. Comparing with the only algorithm capable of doing repeated detection [12], which is centralized and incurs an $O(pn^3)$ time complexity and an $O(pn^2)$ space complexity, our algorithm is superior in performance since d^2 is less than n for any spanning tree with $h > 2$ (essentially any non-centralized configuration). Also, the message complexity of hierarchical detection is significantly lower, as shown in a later section. A comparison between the two algorithms is given in Table I. Notice that $n = d^h$.

Table I
COMPLEXITY COMPARISON BETWEEN HIERARCHICAL DETECTION AND THE CENTRALIZED REPEATED DETECTION ALGORITHM

	Our Hierarchical Algorithm	Centralized Repeated Detection Algorithm [12]
Space Complexity	$O(pn^2)$ (distributed across all processes)	$O(pn^2)$ (at the sink node)
Time Complexity	$O(d^2pn^2)$ (distributed across all processes)	$O(pn^3)$ (at the sink node)
Message Complexity	pn	$p\frac{(d^h-2d)(dh-d-h)-d}{(d-1)^2}$

Contributions of this paper:

1) We present the first decentralized hierarchical algorithm to detect *Definitely(Φ)* in a large-scale distributed system.
2) The hierarchical detection manner of our algorithm makes our algorithm failure-resilient. In our algorithm, each process detects the predicate in the subtree rooted at itself. When a process fails, the detection of the predicate in the remaining processes could be easily resumed because our algorithm has the ability to detect a partial predicate of the global predicate. The same cannot be achieved by the existing centralized or distributed detection algorithms.
3) Hierarchical detection, which is also strongly desirable for large-scale systems, necessarily requires to detect all occurrences of the predicate satisfaction, which we do in our algorithm. None of the existing detection algorithms for *Definitely(Φ)* (except the recent centralized algorithm in [12]) can do such repeated detection.
4) We give a performance analysis of our hierarchical detection algorithm for message , space and time complexity. The result shows that our algorithm is superior to the only known algorithm for repeated detection [12], which is centralized.

The rest of the paper is organized as follows. Section II gives the system model and background on interval-based predicate detection algorithms. Section III presents the hierarchical detection algorithm and its theoretical foundation. Section IV analyses the complexity of the hierarchical detection algorithm. Conclusions are given in Section V.

II. System Model and Background

A. System Model

A distributed system is an undirected graph (P, L), where P is the set of processes and L is the set of communication links. Let $n = |P|$. The n processes asynchronously communicate with each other via the channels in L. We do not assume FIFO channels, thus the messages may be delivered out of order. The execution of process P_i produces a sequence of events $E_i = \langle e_i^0, e_i^1, e_i^2, \cdots \rangle$, where e_i^k is the kth event at process P_i. An event at a process can be message receiving,

Authorized licensed use limited to: University of Illinois at Chicago Library. Downloaded on June 22,2020 at 22:42:12 UTC from IEEE Xplore. Restrictions apply.

message sending, or an internal event. Let $E = \cup_{i \in P} E_i$ denote the set of events executed in a distributed execution. The causal precedence relation between events induces an irreflexive partial order on E. This relation is defined as Lamport's "happens before" relation [13], and denoted as $\prec$. An execution of a distributed system is thus denoted by the tuple $(E, \prec)$.

If the network is a wireless network, each process can communicate only with other processes within communication range. Thus, the topology of such a network cannot be considered as a complete graph, and messages transmitted within such a network usually traverses multiple hops.

We also assume vector clocks [14], [15] are available. Each process maintains a vector clock V of n integers. The vector clock is updated according to the following rules.

1) Before an internal event happens at process P_i, $V_i[i] = V_i[i] + 1$.
2) Before process P_i sends a message, it first executes $V_i[i] = V_i[i] + 1$, then it sends the message piggy-backed with V_i.
3) When a process P_j receives a message with timestamp U from P_i, it executes
$\forall k \in [1 \dots n], V_j[k] = \max(V_j[k], U[k])$;
$V_j[j] = V_j[j] + 1$;
before delivering the message.

The $\prec$ relation between two events can be checked by comparing their corresponding vector clock timestamps, i.e., $e_i \prec e_j \Leftrightarrow V_{e_i} < V_{e_j}$, where $V_{e_i} < V_{e_j}$ means $\forall a \in [1, n], V_{e_i}[a] \leq V_{e_j}[a]$ and $\exists b \in [1, n]$ such that $V_{e_i}[b] < V_{e_j}[b]$. Henceforth, we use the notation $\prec$ between two events and $<$ between their corresponding vector timestamps interchangeably.

B. Background

An interval at a process P_i is the time duration in which the local predicate is true. Due to the lack of synchronized physical clocks at each process, the start and end events of an interval x, denoted as $\min(x)$ and $\max(x)$, respectively, are identified by vector clocks [14], [15]. The detection of either *Possibly*(Φ) or *Definitely*(Φ) is to identify a set of intervals, containing one interval per process in which the local predicate is true, such that a certain condition is satisfied within this set. In [8], [7], [16], it was shown that the conditions to be satisfied for *Possibly*(Φ) or *Definitely*(Φ) to hold within a set X of intervals are as follows:

$$\textit{Possibly}(\Phi): \forall x_i, x_j \in X, \max(x_i) \not\prec \min(x_j) \qquad (1)$$

$$\textit{Definitely}(\Phi): \forall x_i, x_j \in X, \min(x_i) \prec \max(x_j) \qquad (2)$$

Sink process P_1 locally maintains n queues, $Q_1, Q_2, \dots, Q_n$. Whenever a new interval x occurs at some process P_i, P_i sends the vector clock timestamps corresponding to $\min(x)$ and $\max(x)$ to P_1. P_1 then enqueues the interval x onto queue Q_i. By tracking the intervals from all n processes, P_1 checks the heads of all n queues using the condition in (1) or (2) to see whether *Possibly*(Φ) or *Definitely*(Φ) is detected. If any interval is found to violate those conditions, P_1 deletes this interval from its corresponding queue.

III. Hierarchical Detection

A. Basic Idea and Challenges

Our hierarchical detection algorithm works in the following way. We assume a spanning tree is already constructed in the network. This algorithm utilizes this spanning tree to establish a hierarchy for detecting *Definitely*(Φ). Each non-leaf nodes in the tree only maintains queues to track intervals that are sent by its children or that occur locally. Whenever a new interval occurs at a leaf node, it is transmitted to the leaf node's parent which tries to detect the predicate within the subtree rooted at itself. If the predicate is detected in the subtree, the root of the subtree aggregates the set of intervals within which the predicate is detected, and transmits this aggregated interval to its parent. The aggregated interval is treated as a normal interval at the higher levels in the hierarchy, and is used for detecting the predicate within an even larger subtree. Once an aggregated interval is sent to the parent (higher level process), the parent detects occurrences of the predicate within the larger subtree rooted at itself using aggregated intervals received from its children, and generates the aggregated intervals for its level once a satisfaction of the predicate is detected. Whenver the predicate is detected at some subtree, the root of that subtree will perform the operations necessary for doing repeated detection within that subtree. The same procedure repeats at each level of the hierarchy. At the root of the spanning tree, the predicate is detected for the entire network.

From the above description, we can see that the difficulties in realizing this algorithm are: (i) how to aggregate a set of intervals, and (ii) how to do repeated detection of *Definitely*(Φ) using aggregated intervals from a lower level in the hierarchy.

In [7], the authors outlined an approach to do hierarchical detection of *Definitely*(Φ) by trying to address (i) above. However, their solution lacks in the following 2 aspects.

1) In [7], the authors assumed a specific partial order in a set of intervals where *Definitely*(Φ) is detected. This

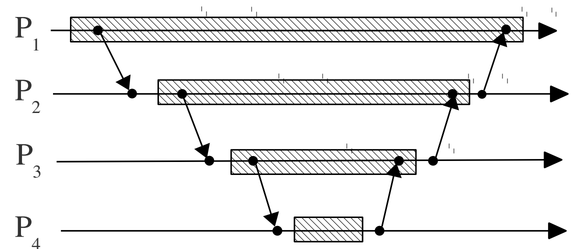

Figure 1. The approach in [7] works only if the intervals are nested.

Authorized licensed use limited to: University of Illinois at Chicago Library. Downloaded on June 22,2020 at 22:42:12 UTC from IEEE Xplore. Restrictions apply.

partial order requires that the intervals in the set can be ordered into $x_1, x_2, \ldots, x_k$ such that $\forall i, j \in [1, k]$, if $i < j$ then $\min(x_i) \prec \min(x_j) \wedge \max(x_j) \prec \max(x_i)$. As a result, their solution requires the set of intervals within which *Definitely*(Φ) is detected to be nested, meaning that the intervals in the set establish a relation as shown in Figure 1. However, the relation shown in Figure 1 will not always hold in a set of intervals satisfying *Definitely*(Φ), as we will show in Figure 3.

2) The approach outlined in [7] does not do repeated detection. Being able to detect all occurrences of the predicate at each level is essential to hierarchical detection. This statement is justified using Figure 2.

We assume the hierarchy is formed as shown in Figure 2(a). From Figure 2(b), we can observe that the first set of intervals detected at P_2 satisfying *Definitely*(Φ) consists of x_1 and x_2, and its aggregation will be sent to the process in the higher level, i.e., P_3. In addition to receiving this solution set, after P_3 receives interval x_5 from P_4 and interval x_4 occurs at P_3, P_3 will start the detection at the higher level. However, *Definitely*(Φ) cannot be detected in the set $\{x_1, x_2, x_4, x_5\}$. If only a one-time detection algorithm runs at P_2, which is the case in the approach in [7], then the only set of intervals P_2 ever reports to P_3 is $\{x_1, x_2\}$ and the later occurrence of the predicate for P_1 and P_2 in the set $\{x_1, x_3\}$ will be ignored. Therefore, the set $\{x_1, x_3, x_4, x_5\}$ within which the predicate could be detected for all 4 processes

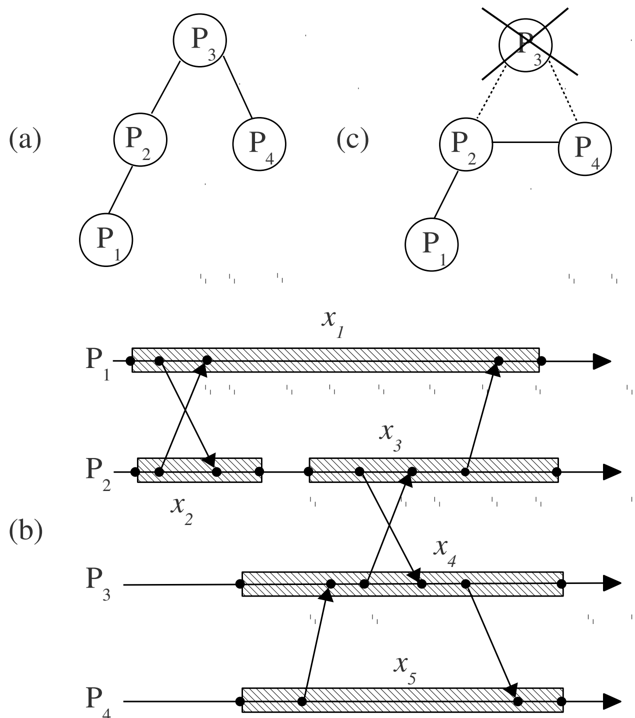

Figure 2. (a) The spanning tree consists of 4 processes. (b) Timing diagram showing the relation between intervals from those 4 processes. (c) The spanning tree reconnected after a node failure happens at process P_3.

will never be detected by P_3. This example shows that being able to find *all occurrences* of the predicate at *each level* is necessary to the hierarchical detection algorithm.

Without a proper way to aggregate intervals and without a way to repeatedly detect predicates, the approach given in [7] will fail to detect the predicates at the intermediate nodes as well as at the top of the hierarchy.

B. Example Scenario of Our Algorithm

In this subsection, still using Figure 2, we show how our algorithm handles this example scenario.

When *Definitely*(Φ) is first detected in $\{x_1, x_2\}$ at P_2 for processes P_1 and P_2, our algorithm will identify one interval from this set such that it will never form part of a future solution set detected by P_2. After identifying such an interval, in this case x_2, P_2 will remove x_2 from its corresponding queue after sending the aggregated interval of set $\{x_1, x_2\}$ to P_3. P_2 then continues the detection for later occurrences of the predicate. When interval x_3 finishes, P_2 will detect a second occurrence of the predicate within the subtree rooted at P_2 in the set $\{x_1, x_3\}$. P_2 sends another aggregated interval of this set to P_3. At process P_3, after local interval x_4 finishes and P_3 receives the aggregation of $\{x_1, x_2\}$ from P_2 and the interval x_5 from P_4, P_3 will attempt to detect the predicate at its level. The first attempt will fail, since the set $\{x_1, x_2, x_4, x_5\}$ does not satisfy *Definitely*(Φ). As part of this failed attempt, P_3 will remove the aggregation of $\{x_1, x_2\}$ from its queue. When P_3 receives the aggregation of $\{x_1, x_3\}$ from P_2, a second attempt to detect the predicate begins. This time, the predicate is detected in the set $\{x_1, x_3, x_4, x_5\}$. Thus the predicate is detected for all 4 processes.

If P_3 fails after interval x_4 finishes, our algorithm will reconnect the spanning tree. The resulting tree could be the one shown in Figure 2(c). In this case, P_2 will report its later aggregated interval of the set $\{x_1, x_3\}$ to its new parent, P_4. P_4 will still be able to detect the predicate in the set $\{x_1, x_3, x_5\}$ for the remaining processes P_1, P_2 and P_4. The failure of P_3 will not affect the detection of the predicate in the remaining processes because what is lost is only the local intervals from P_3. In contrast, the only other repeated detection algorithm [12], which is centralized, will fail to detect future occurrences of the predicate if the sink fails because all intervals sent to the sink will be lost.

From this example, we can observe that the key aspects of our hierarchical detection algorithm lie in

1) the way to aggregate a solution set, and
2) the way to identify at least one interval from a solution set for removal to ensure progress for repeated detection

at *each level* in the hierarchy. In the rest of this section, we will show how we solve these problems.

Authorized licensed use limited to: University of Illinois at Chicago Library. Downloaded on June 22,2020 at 22:42:12 UTC from IEEE Xplore. Restrictions apply.

C. Aggregation of Intervals to Detect *Definitely*(Φ)

In [7], [16], it was shown that for *Definitely*(Φ) to hold in a set X of intervals, the following needs to be true

$$\forall x_i, x_j \in X, \min(x_i) < \max(x_j)$$

This property was named as *overlap*(X). Our objective is to decentralize the detection of *Definitely*(Φ). We first consider the scenario where *Definitely*(Φ) has been detected in each of the two sets of intervals X and Y and we want to detect *Definitely*(Φ) in $X \cup Y$.

Assume now, we have 4 processes in the network with their timing diagram shown in Figure 3(a). The intervals occurring at each process are shaded. The vector clock timestamps identifying the lower and higher bound of each interval are also illustrated in the figure. Intervals x_1 from process P_1 and x_2 from process P_3 form set X, while intervals y_1 and y_2 from process P_2 and P_4, respectively, form set Y. It can be checked that both *overlap*(X) and *overlap*(Y) are true.

Now, to show that *Definitely*(Φ) is also detected in all 4 processes, or equivalently *overlap*($X \cup Y$), we need to show

$$\forall i, j \in \{1, 2\}, \min(x_i) < \max(y_j) \wedge \min(y_j) < \max(x_i) \tag{3}$$

From Figure 3(b), we can observe that, if we take the component-wise maximum of $\min(x_1)$ and $\min(x_2)$ (illustrated in bold) to form a new vector u, then the first conjunct in Eq. (3) is equivalent to

$$\forall j \in \{1, 2\}, u < \max(y_j) \tag{4}$$

Furthermore, if we take the component-wise minimum of $\max(y_1)$ and $\max(y_2)$ (illustrated in underline) to form another new vector r, then Eq. (4) is equivalent to $u < r$. The same operations can also be applied to show the second conjunct of Eq. (3) using aggregated vectors v and q.

This gives the inspiration to aggregate a set of intervals into a single one to detect *Definitely*(Φ) in a larger set of intervals. For sets X and Y in Figure 3, their aggregated intervals are denoted as $\sqcap(X)$ and $\sqcap(Y)$, respectively. The way to aggregate those two sets using component-wise minimum or maximum is shown in Figure 3(b).

Formally, for an arbitrary set X of intervals, for which *overlap*(X) is true, we define an aggregation function $\sqcap(X)$ of intervals in X, in terms of vector timestamps, as:

$$\forall i \in [1, n], \min(\sqcap(X))[i] = \max_{x \in X}(\min(x)[i]) \tag{5}$$

$$\forall i \in [1, n], \max(\sqcap(X))[i] = \min_{x \in X}(\max(x)[i]) \tag{6}$$

With this formal definition of the aggregation function $\sqcap$, we show the following theorem.

Theorem 1. *Let X, Y and Z be sets of intervals, such that $Z = X \cup Y$. Then $overlap(Z)$ iff $overlap(X) \wedge overlap(Y) \wedge overlap(\sqcap(X), \sqcap(Y))$.*

Proof: ($\Rightarrow$) $overlap(X)$ and $overlap(Y)$ are clearly true since $X, Y \subseteq Z$. Now consider an interval $y \in Y$. Since $overlap(Z)$, $\forall x \in X, \min(x) < \max(y)$. Thus $\min(\sqcap(X)) < \max(y)$. Since this is true for all $y \in Y$, $\min(\sqcap(X)) < \max(\sqcap(Y))$. The same deduction applies to $\min(\sqcap(Y)) < \max(\sqcap(X))$. So, we have $overlap(\sqcap(X), \sqcap(Y))$.

($\Leftarrow$) From $overlap(\sqcap(X), \sqcap(Y))$ we have $\min(\sqcap(X)) < \max(\sqcap(Y)) \wedge \min(\sqcap(Y)) < \min(\sqcap(X))$. For any interval $x \in X$, we have $\min(x) < \min(\sqcap(X))$. For any interval $y \in Y$, we have $\max(\sqcap(Y)) < \max(y)$. Since $\min(\sqcap(X)) < \max(\sqcap(Y))$, we have for any $x \in X$ and any $y \in Y$, $\min(x) < \max(y)$. Similarly, we can deduce that for any $x \in X$ and any $y \in Y$, $\min(y) < \max(x)$. Since we already have $overlap(X)$ and $overlap(Y)$, now we have $overlap(Z)$. ■

Theorem 1 shows that we can aggregate a set of intervals X into a single interval $\sqcap(X)$ which can represent the entire set in detecting *Definitely*(Φ) within an even larger set of intervals. $\sqcap(X)$ is uniquely identified by $\min(\sqcap(X))$ and $\max(\sqcap(X))$. These are not events but cuts in execution $(E, \prec)$, identified by their vector timestamps.

Theorem 1 only covers the scenario of two sets of intervals and their union. In the spanning tree, some processes may have more than 2 children. Below, we extend Theorem 1 to scenarios involving more than two sets of intervals.

Lemma 1. *Let $X_1, X_2, \ldots, X_d$ be d sets of intervals, and Z be the union of all d sets. Thus $Z = \cup_{i=1}^{d} X_i$. Then $overlap(Z)$ iff $\wedge_{i=1}^{d} overlap(X_i) \wedge overlap(\sqcap(X_1), \sqcap(X_2), \ldots, \sqcap(X_d))$*

Proof: ($\Rightarrow$) $\wedge_{i=1}^{d} overlap(X_i)$ is clearly true since $X_i \subset Z$. Since $overlap(Z)$, we have $\forall i, j \in [1, d], overlap(X_i \cup X_j)$. Thus, according to Theorem 1, we have $\forall i, j \in [1, d], overlap(\sqcap(X_i), \sqcap(X_j))$. This means, $\forall i, j \in [1, d], \min(\sqcap(X_i)) < \max(\sqcap(X_j))$. Thus, we have $overlap(\sqcap(X_1), \sqcap(X_2), \ldots, \sqcap(X_d))$

($\Leftarrow$) Since $\wedge_{i=1}^{d} overlap(X_i) \wedge overlap(\sqcap(X_1), \sqcap(X_2), \ldots, \sqcap(X_d))$, we have $\forall i, j \in [1, d], overlap(X_i \cup X_j)$. This means, by picking any two intervals y_1, y_2 from Z, it is always true that $\min(y_1) < \max(y_2)$. This is because there will always be a pair of $i, j \in [1, d]$, such that $y_1 \in X_i$ and $y_2 \in X_j$. So, we have $overlap(Z)$. ■

For our hierarchical detection algorithm, each process P_i in the spanning tree detects *Definitely*(Φ) within the subtree rooted at itself. Once the predicate is detected, P_i aggregates the set of intervals within which the predicate is detected using $\sqcap$ and sends the aggregated interval to its parent. At higher levels in the spanning tree, the predicate within the subtree will be detected based on aggregated intervals received from children processes. Lemma 1 ensures that, by testing the *overlap* property on the aggregated intervals, the predicate can be detected within a larger set of intervals. At higher levels, the aggregation function will also be applied

Authorized licensed use limited to: University of Illinois at Chicago Library. Downloaded on June 22,2020 at 22:42:12 UTC from IEEE Xplore. Restrictions apply.

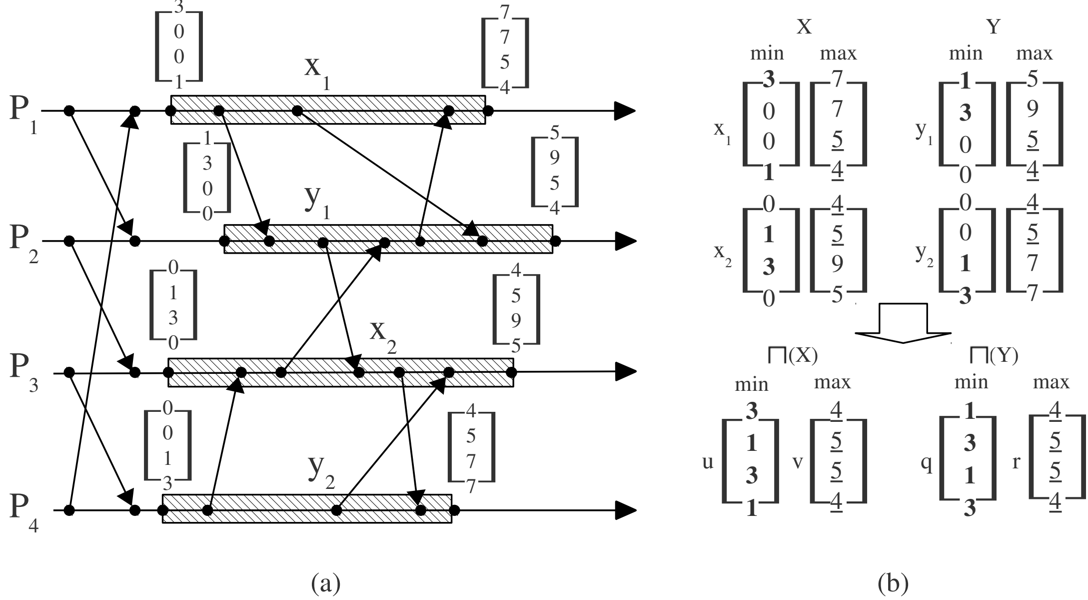

Figure 3. Example showing the aggregation of intervals for detecting *Definitely(*Φ*)*. (a) The timing diagram of the system is given. An interval from each process is marked in shade along with the vector clock timestamps identifying the lower and higher bounds. (b) The two sets of intervals X and Y consisting of intervals from (a) are shown. The way to aggregate each set is also illustrated. Component-wise maximums among all lower bounds in the same set are marked in bold while the component-wise minimums among all higher bounds in the same set are marked in underline.

to the aggregated intervals. However, we notice that, for two sets of intervals X and Y,

$$\sqcap(\sqcap(X), \sqcap(Y)) = \sqcap(X \cup Y) \quad (7)$$

So, applying the aggregation function on aggregated intervals is equivalent to applying it on the union of all sets.

D. Repeated Detection

In [12], the author showed how repeated detection can be done in the centralized *Definitely(*Φ*)* detection algorithm. Basically, repeated detection requires identifying a certain interval from a solution set such that this single interval cannot be part of a future solution set, and then removing this interval from the corresponding queue.

Doing the same in the hierarchical detection algorithm is more complex. In the hierarchical algorithm, the detection takes place at each level. At higher levels, the solution set consists of both aggregated intervals and non-aggregated intervals. Each aggregated interval represents a solution set at the lower level. Identifying a certain interval for removal now is to identify a solution set that cannot be part of a future solution at a higher level, and removing an aggregated interval x in the solution set will remove all the intervals aggregated by x. This is very different from the situation in the centralized algorithm in which the sink only needs to consider non-aggregated intervals. Below, we show how repeated detection can be done in the hierarchical detection algorithm.

First, for aggregated intervals generated at the same process, we have

Theorem 2. *For an aggregated interval $\sqcap(X)$ generated at process P_a and a later aggregated interval $\sqcap(X')$ generated at the same process,* $\min(\sqcap(X)) < \max(\sqcap(X)) < \min(\sqcap(X')) < \max(\sqcap(X'))$.

Proof: Since $\sqcap(X)$ is an aggregated interval, the set of intervals X it aggregates satisfy the condition $overlap(X)$. Thus $\forall x_i, x_j \in X, \min(x_i) < \max(x_j)$. Also, according to the definition in Eq. (5), we know that the elements in $\min(\sqcap(X))$ and $\max(\sqcap(X))$ are equal to the component-wise maximum or minimum among all $\min(x_i)$ and $\max(x_i)$, respectively. Since $\forall x_i, x_j \in X, \min(x_i) < \max(x_j)$, we have $\forall x_i, x_j \in X, \forall l \in [1, n], \min(x_i)[l] \leq \max(x_j)[l]$. Thus, $\forall l \in [1, n], \min(\sqcap(X))[l] \leq \max(\sqcap(X))[l]$. So, $\min(\sqcap(X)) < \max(\sqcap(X))$. The same can also be shown for $\sqcap(X')$.

Since $\sqcap(X')$ is generated after $\sqcap(X)$, it means X' is a solution set within the subtree rooted at P_a that occurs after the solution set X. Thus, there exists at least one interval x'_b in X', such that x'_b occurs after the corresponding interval x_b in X which comes from the same process. So, we have $\max(x_b) < \min(x'_b)$. Also, according to the definition in Eq. (5), we know that $\forall x_i \in X, \forall x'_i \in X', \max(\sqcap(X)) < \max(x_i) \wedge \min(x'_i) < \min(\sqcap(X'))$. Thus, $\max(\sqcap(X)) < \min(\sqcap(X'))$. ■

For any two intervals x and x' that occur (local intervals) or are generated (aggregated intervals) at the same process, if $\max(x) < \min(x')$, we call x' a successor of x and denote it as $succ(x)$. Theorems 1 and 2 prove that the aggregated intervals are treated just as the non-aggregated intervals at the higher levels in the hierarchy. Now we show how we can identify an interval, aggregated or not, from a solution

Authorized licensed use limited to: University of Illinois at Chicago Library. Downloaded on June 22,2020 at 22:42:12 UTC from IEEE Xplore. Restrictions apply.

set such that it can be safely removed.

In order for an interval x_i in a solution set to be part of a future solution set, there needs to be at least one interval x_j from the same solution set, such that $overlap(x_i, succ(x_j))$ is true. From [12], we know that this is equivalent to

$$\min(succ(x_j)) < \max(x_i) \quad (8)$$

Then, if for all interval $x_j (j \neq i)$ from the solution set X, Eq. (8) is false, we have that $overlap(x_i, succ(x_j))$ is false for all $x_j (j \neq i)$ from the solution set. Thus x_i can be safely removed from the head of the queue. So, we have

$$\text{remove } x_i \text{ iff} \\ \forall x_j \in X(j \neq i), \min(succ(x_j)) \nless \max(x_i) \quad (9)$$

Since $\max(x_j) < \min(succ(x_j))$, from [12], we know the test condition in Eq. (9) can be approximated to

$$\text{remove } x_i \text{ iff } \forall x_j \in X(j \neq i), \max(x_j) \nless \max(x_i) \quad (10)$$

Since we do not know the values in $\min(succ(x_j))$ until that interval gets reported from the lower level, in order to identify the interval for removal as soon as possible, the approximated condition in Eq. (10) is what we use to prune the queues. Although it is only an approximation, we now show that it is actually correct and capable of always identifying at least one interval for removal.

Theorem 3. *(Safety) Once a solution set X is detected at any process in the hierarchy, only intervals $x_i \in X$ (x_i may be aggregated or not) that cannot be part of another solution are removed from their queues.*

Proof: Since Eq. (10) ⇒ Eq. (9), any interval removed using the condition in Eq. (10) will also satisfy the condition in Eq. (9). Thus, those intervals cannot be part of any future solution set. Therefore, even if Eq. (9) is only an approximation, it still guarantees safety. ■

Theorem 4. *(Liveness) For any solution set X detected at any process in the hierarchy, at least one interval (aggregated or not) gets removed from its queue.*

Proof: Assume that the condition in Eq. (10) cannot be satisfied by some solution set X. Then, it means that for all intervals $x_i \in X$, aggregated or not, there exists another interval $x_j \in X$, such that $\max(x_j) < \max(x_i)$. This condition will eventually cause one interval x_k to satisfy $\max(x_k) < \max(x_k)$, which is impossible. So the assumption is false, and thus the condition in Eq. (9) holds for any solution set. Thus, Eq. (9) guarantees liveness. ■

With the safety and liveness of the condition in Eq. (10) proved, we can safely use it to prune the queues so that future occurrences of the predicate at each level in the hierarchy can be repeatedly detected.

E. Hierarchical Detection Algorithm

With Theorems 1, 3 and 4, we have the theoretical foundation for the hierarchical detection algorithm we outlined in Section III-A. The algorithm is listed in Algorithm 1. Each process in the spanning tree tracks the intervals occurring locally and those sent from its children. The intervals sent from the child process can be non-aggregated intervals or aggregated ones, depending on whether the child is a leaf node. By checking the intervals received in the queues (Lines

Algorithm 1 Hierarchical decentralized detection of conjunctive definitely predicates, adapted from [12] (Code for P_i)

```
number of children: l
queue for P_i: Q_0 ← ⊥
queues for children: Q_1, Q_2, ..., Q_l ← ⊥
set of int: updatedQueues, newUpdated ← {}
int: count
On receiving an interval from child P_j at P_i:
1.  Enqueue the interval onto queue Q_j;
2.  if (number of intervals on Q_j is 1) then
3.    updatedQueues = {j};
4.    while (updatedQueues is not empty)
5.      newUpdated = {};
6.      for each a ∈ updatedQueues do
7.        if (Q_a is not empty) then
8.          x = head of Q_a;
9.          for b = 0...l(b ≠ a) do
10.           if (Q_b is not empty) then
11.             y = head of Q_b;
12.             if (min(x) ≮ max(y)) then
13.               add b to newUpdated;
14.             if (min(y) ≮ max(x)) then
15.               add a to newUpdated;
16.     Delete heads of all Q_c where c ∈ newUpdated;
17.     updatedQueues = newUpdated;
18. if (all queues are non-empty) then
19.   if (P_i has parent in the spanning tree)    // non-root
20.     report ⊓(heads of all queues) to parent;
21.   else    // root
22.     report predicate detected.
23.   for (a = 0...l) do
24.     count = 0;
25.     for (b = 0...l(b ≠ a)) do
26.       for (c = 1...n) do
27.         if (max(head(Q_a))[c] < max(head(Q_b))[c]) then
28.           count++;
29.           break;
30.     if (count = l) then
31.       add a to newUpdated;
32.   Delete heads of all Q_a where a ∈ newUpdated;
33.   updatedQueues = newUpdated;
```


Authorized licensed use limited to: University of Illinois at Chicago Library. Downloaded on June 22,2020 at 22:42:12 UTC from IEEE Xplore. Restrictions apply.

(1)-(17)), each process attempts to detect the predicate within the subtree rooted at itself. Once a solution set is found (Lines (18)), the root of the subtree aggregates the set and sends it to its parent (Lines (19)-(20)). At the higher level in the hierarchy, the parent determines if the predicate can be detected in an even larger subtree rooted at itself by repeating the same detection procedure (Lines (1)-(17)). When the root of the spanning detects a solution set, a satisfaction of the predicate is detected within the whole network (Lines (21)-(22)).

Each time the predicate is detected at some process, Lines (23)-(33) prune the heads of the queues so that future occurrences of the predicate at the same level can be repeatedly detected. For each interval x_i in the solution set X, this procedure checks x_i against all other intervals x_j in X to see if $\forall x_j \in X(j \neq i), \max(x_j) \not< \max(x_i)$. Each time an interval x_i is to be checked, a counter is initialized to 0. For each interval $x_j(j \neq i)$, if $\max(x_j) \not< \max(x_i)$ then the counter is increased by 1. After x_i is checked against all other intervals x_j, if the counter equals l, which is the total number of intervals in the solution set X minus 1, then interval x_i satisfies the condition in Eq. (10). Thus, we can safely remove x_i from the corresponding queue. In Algorithm 1, the intervals to be processed can be aggregated intervals. Thus when comparing the vector timestamps of two intervals (Lines 12, 14, 26-27), we cannot compare them in $O(1)$ time as we can do with the normal intervals. This will affect the time complexity of this algorithm, as we will show in Section IV-C.

Although Algorithm 1 has the same basic structure as the centralized algorithm given in [12], it is essentially different. Algorithm 1 detects *Definitely(*Φ*)* in a hierarchical manner and performs tests on aggregated intervals. Instead of one central server process maintaining n queues, each process in Algorithm 1 maintains queues only for itself and its children in the spanning tree. When the predicate is detected at non-root processes, the solution set is aggregated for processes in the higher level to detect the predicate in a larger area.

F. Dealing with Node Failures

When process P_i fails, the local queue in P_i's parent P_j corresponding to P_i will not be updated further. Also, subtrees rooted at P_i's children will be disconnected from the spanning tree. Thus, a mechanism to detect node failures in the spanning tree is needed. This can be achieved by assuming that each process in the spanning tree sends heart-beat messages to its parent and children. So, when a process P_i fails, both its parent and children will stop receiving heart-beat messages from P_i and know about P_i's failure.

P_i's parent P_j will remove its local queue corresponding to P_i and keep detecting the predicate with the aggregated intervals from the remaining children. Each subtree rooted at each of P_i's children will reconnect itself to the system-wide spanning tree by establishing a link between a node in the subtree and its neighbor which is still in the spanning tree. After this topology change, nodes having new child processes will create a new local queue to receive aggregated intervals reported from each new child and detect predicate within the new subtree. Nodes that lose children processes will remove the corresponding local queues. This is illustrated in Section III-B and Figure 2(a), (c).

Although these two operations change the spanning tree each time a node fails, due to the fact that the detection of the predicate happens at each level in the spanning tree, the failure of P_i will not affect the continuation of the monitoring of the predicate within the rest of the network and Theorems 1, 3 and 4 ensure the correctness of further detection of the predicate within the remaining processes.

IV. COMPLEXITY

Three metrics: space complexity, time complexity and message complexity, are used in the complexity analysis, which is done in terms of the following parameters:

- n: the number of nodes in the network
- p: the maximum number of intervals per process
- d: the maximum number of children any process in the spanning tree can have
- h: the height of the spanning tree
- α: the probability that intervals from d children overlap and can be aggregated as per Eq. (5) at one higher level.

Table I summarized the results. Notice that $n = d^h$.

A. Message Complexity

In the hierarchical detection algorithm, the messages are transmitted along the edges of the spanning tree. For the leaf nodes in the spanning tree, each time an interval occurs locally, it sends this interval to its parent. A non-leaf node only sends one aggregated interval to its parent once it detects the predicate within the subtree rooted at itself.

At a leaf node (level 1), all intervals occurring locally are sent to its parent. At level i, the number of aggregated intervals generated by a single process is $d\alpha$ times the number of intervals received from any child in level $i-1$. This can be justified using the reasoning in [12]. Each time an aggregated interval is generated at a certain process, the predicate is detected within the corresponding subtree.

With the above analysis, we know that at level i, $\alpha^{i-1}d^{i-1}p$ number of aggregated intervals will be sent to level $i+1$. Then, we can derive the total number of messages transmitted in the system for Algorithm 1. For a spanning tree of degree d and height h,

$$\begin{aligned} \text{total \# of msgs} &= \sum_{i=1}^{h-1} d^{h-i} p d^{i-1} \alpha^{i-1} \\ &= p d^{h-1} \frac{1-\alpha^{h-1}}{1-\alpha} \end{aligned} \quad (11)$$

We now compare with the message complexity of the centralized repeated detection algorithm [12]. Notice that each

Authorized licensed use limited to: University of Illinois at Chicago Library. Downloaded on June 22,2020 at 22:42:12 UTC from IEEE Xplore. Restrictions apply.

message in hierarchical detection is transmitted only 1 hop, always to the immediate parent. For a message that traverses h hops in a wired or wireless network, it is equivalent to h point-to-point messages, since the communication channels are occupied h times. Thus, when running the centralized repeated detection algorithm in the same network where the sink collects intervals from the other processes via a spanning tree of degree d and height h, we need to account for the cost coming from each message having to traverse several hops to reach the sink.

In the centralized algorithm [12], messages sent from level i need to traverse $h-i$ hops to reach the sink. Furthermore, each local interval needs to be transmitted all the way to the sink. So at level i, across all nodes at that level in the spanning tree, $g(i) = pd^{h-i}(h-i)$ messages are generated by the centralized algorithm. We can thus deduce the total number of messages generated by the centralized repeated detection algorithm [12].

$$\text{total \# of msgs} = \sum_{i=1}^{h-1} pd^{h-i}(h-i) = ph\sum_{i=1}^{h-1} d^{h-i} - p\sum_{i=1}^{h-1} id^{h-i} \quad (12)$$

Let $k = \sum_{i=1}^{h-1} id^{h-i}$, then

$$dk = \sum_{i=1}^{h-1} id^{h-i+1}$$

$$(d-1)k = \sum_{i=2}^{h} d^i + (h-1)d = \frac{d^2(1-d^{h-1})}{1-d} + (h-1)d$$

$$k = \frac{d^{h+1} + d^2h - 2d^2 - dh + d}{(d-1)^2} \quad (13)$$

Substituting Eq. (13) into Eq. (12), we have

$$\text{total \# of msgs} = p\frac{(d^h - 2d)(dh - d - h) - d}{(d-1)^2} \quad (14)$$

Figures 4 and 5 compare the message complexity between Algorithm 1 (Eq. (11)) and the centralized repeated detection algorithm [12] (Eq. (14)) with different parameters. In Figure 4, $d = 2$ and α is set to 0.1 and 0.45. In Figure 5, α takes the same values while d is set to 4. From these two graphs, we can observe that for the same p, the height h and degree d of the spanning tree, or equivalently the size of the network, impacts the total number of messages. Also, with a smaller α, the number of messages decreases. Furthermore, observe that p is a linear factor in both Eq. (11) and Eq. (14). So, if we fix other parameters, as p increases, the total number of messages also increases linearly. As a conclusion, we can see that the hierarchical detection algorithm has a better message complexity comparing to the centralized repeated detection algorithm, especially when the system is large-scale.

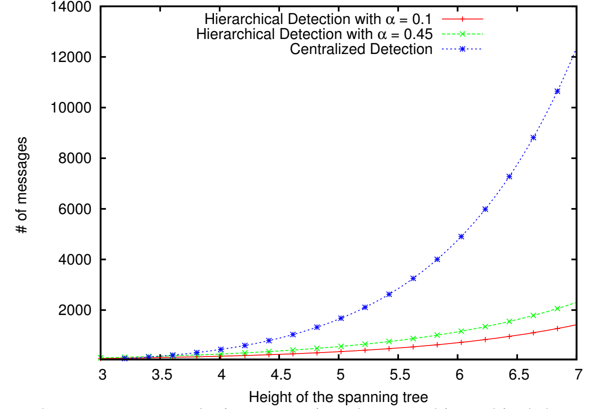

Figure 4. Message complexity comparison between hierarchical detection and centralized detection, with $d = 2, p = 20$.

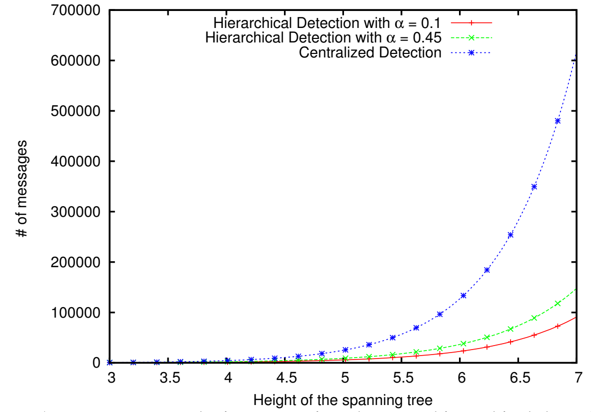

Figure 5. Message complexity comparison between hierarchical detection and centralized detection, with $d = 4, p = 20$.

B. Space Complexity

In hierarchical detection, each process other than the leaf nodes only needs to maintain $O(d)$ queues. The number of intervals in the local queue Q_0 is $O(p)$. Also, each process needs to store the aggregate intervals from its $O(d)$ children.

In the worst case, the space complexity is the number of intervals in all the nodes. Eq. (11) gives the total number of aggregated intervals. Observe that $n = d^h$. Thus, the total number of aggregated intervals is affected by $\frac{1-\alpha^{h-1}}{1-\alpha}$, which is $\sum_{i=1}^{h-1} \alpha^{i-1}$ and is bounded by $h-1$. Since $n = d^h$, we know $\frac{1-\alpha^{h-1}}{1-\alpha}$ is $O(\log(n))$ in the worst case.

Furthermore, for a particular predicate *Definitely*(Φ) to be detected in a network, α can be treated as a constant since α is only related to the predicate to be detected and the execution of the distributed system. Thus $\frac{1-\alpha^{h-1}}{1-\alpha}$ is $O(1)$. $\frac{1-\alpha^{h-1}}{1-\alpha}$ only becomes large when $\alpha \to 1$ and h is very large. This means that almost all attempts to detect the predicate at every level of the hierarchy will succeed in a large-scale network. This is an impractical assumption. For a practical value of α, even as large as 0.5, $\frac{1-\alpha^{h-1}}{1-\alpha}$ is still less than

Authorized licensed use limited to: University of Illinois at Chicago Library. Downloaded on June 22,2020 at 22:42:12 UTC from IEEE Xplore. Restrictions apply.

or equal to 2 in this particular case. For impractical values of α, $\frac{1-\alpha^{h-1}}{1-\alpha}$ will be bounded by $\min(h, \frac{1}{1-\alpha})$. Thus, we assume that the total number of aggregated intervals stored in all processes is $O(pn)$.

In addition, each process will store its local $O(p)$ intervals, and that is an additional $O(pn)$ intervals across the whole network. Since the storage size of both regular intervals and aggregated intervals is $O(n)$, the storage cost of the hierarchical detection algorithm is $O(pn^2)$, distributed across all the nodes in the network.

Comparing with the centralized repeated detection algorithm [12], which incurs an $O(pn^2)$ storage cost at the sink/root of the spanning tree, hierarchical detection algorithm does not place all the storage cost at a single process, which makes it suitable for large-scale systems where a single process cannot afford storing all the data.

C. Time Complexity

From the previous subsection, we know that there are $O(pn)$ aggregated intervals and $O(pn)$ non-aggregated intervals stored across all nodes in the network. When running Algorithm 1, each node needs to check all intervals stored locally (Lines (1)-(22)) to detect *Definitely*(Φ). Since the total number of intervals is $O(pn)$, and each interval will be compared with $O(d)$ other intervals with each comparison taking $O(n)$ time, Lines (1)-(22) in Algorithm 1 will incur an $O(dn^2p)$ time complexity distributed across all the nodes in the network. For Lines (23)-(33), each time the predicate is detected at some node, this part of the code will be executed. Since Lines (23)-(33) compare the heads of $O(d)$ queues, each time Lines (23)-(33) execute, they will take $O(d^2n)$ time. Since the total number of aggregated intervals is $O(pn)$ across all nodes, the maximum number of times the predicate can be detected across all levels in the spanning tree is $O(pn)$. Thus, Lines (23)-(33) incurs an $O(d^2n^2p)$ time complexity across all iterations and all nodes in the network. So, in total, the time complexity of the hierarchical detection algorithm is $O(d^2n^2p)$, spread across all n nodes.

In a large-scale network running the hierarchical detection algorithm, $h > 2$, otherwise the algorithm becomes a centralized algorithm. Since $n = d^h$, we infer that $n > d^2$. Thus, comparing the hierarchical detection algorithm with the centralized repeated detection algorithm, which incurs an $O(pn^3)$ time complexity, the hierarchical detection algorithm has a lower time complexity, especially when h is large. Furthermore, this time complexity is distributed across all nodes, which is not the case in the centralized algorithm.

V. Conclusions

In this paper, we proposed the first decentralized hierarchical algorithm that repeatedly detects all occurrences of *Definitely*(Φ) for a conjunctive predicate Φ. Such an algorithm is essential for large-scale systems where individual nodes have limited computation resources and are subject to failure. Our algorithm detects the predicate at each level in the hierarchy, and thus is able to detect a partial predicate of the global predicate. This enables our algorithm to easily resume the detection after a node failure. Furthermore, comparing with the only other algorithm capable of doing repeated detection [12], our algorithm distributes a lower time cost, and the same space cost, across all processes in the network, and reduces the number of control messages significantly.

References

[1] M. Shen, A. Kshemkalyani, and A. Khokhar, "Detecting tree distributed predicates," *2012 41st International Conference on Parallel Processing Workshops (ICPPW)*, pp. 598–599, 2012.

[2] M. D. Rosa, S. Goldstein, P. Lee, P. Pillai, and J. Campbell, "Programming modular robots with locally distributed predicates," *Proceedings of the IEEE ICRA*, 2008.

[3] M. D. Rosa, S. Goldstein, J. C. P. Lee, and P. Pillai, "Detecting locally distributed predicates," *ACM Transactions on Autonomous and Adaptive Systems*, June 2011.

[4] A. Kshemkalyani and M. Singhal, *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, 2008.

[5] R. Cooper and K. Marzullo, "Consistent detection of global predicates," *Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging*, pp. 163–173, 1991.

[6] ——, "Consistent detection of global predicates," *ACM SIGPLAN Notices. Vol. 26.*, pp. 167–174, 1991.

[7] V. K. Garg and B. Waldecker, "Detection of strong unstable predicates in distributed programs," *IEEE Transactions on Parallel & Distributed Systems 7, 12*, pp. 1323–1333, 1996.

[8] ——, "Detection of weak unstable predicates in distributed programs," *IEEE Transactions on Parallel & Distributed Systems 5, 3*, pp. 299–307, 1994.

[9] V. Garg and C. Chase, "Distributed algorithms for detecting conjunctive predicates," *Proc. 15th IEEE ICDCS*, pp. 423–430, 1995.

[10] M. Hurfin, M. Mizuno, M. Raynal, and M. Singhal, "Efficient distributed detection of conjunctions of local predicates," *IEEE Trans. Software Engineering, 24, 8*, pp. 664–677, 1998.

[11] P. Chandra and A. Kshemkalyani, "Distributed algorithm to detect strong conjunctive predicates," *Information Processing Letters, 87, 5*, pp. 243–249, 2003.

[12] A. Kshemkalyani, "Repeated detection of conjunctive predicates in distributed executions," *Information Processing Letters, 111, 9*, pp. 447–452, 2011.

[13] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM 21, 7*, pp. 558–565, 1978.

[14] F. Mattern, "Virtual time and global states of distributed systems," *Proceedings of the Parallel and Distributed Algorithms Conference*, pp. 215–226, 1988.

[15] C. Fidge, "Logical time in distributed computing systems," *IEEE Computer*, pp. 28–33, Aug, 1991.

[16] A. Kshemkalyani, "Temporal interactions of intervals in distributed systems," *Journal of Computer and System Sciences, 52, 2*, pp. 287–298, 1996.

Authorized licensed use limited to: University of Illinois at Chicago Library. Downloaded on June 22,2020 at 22:42:12 UTC from IEEE Xplore. Restrictions apply.