局所的トリガ数え上げ問題に対する分散アルゴリズム

安達駿，大下福仁，角川裕次，増澤利光
大阪大学情報科学科研究科

概要

ネットワークで接続された多数のプロセッサから構成される分散システムを利用した，外部環境モニタリングを考える．プロセッサに付随したセンサが降雨や渋滞などのイベントを検知すると，プロセッサでトリガが発生する．このとき，システム全体で発生したトリガの総数がある閾値に達したことを検出する問題はトリガ数え上げ問題とよばれ，これまでにいくつかの分散アルゴリズムが提案されている．本報告では，各プロセッサの近傍に注目し，近傍プロセッサで発生したトリガの総数がある閾値に達したことを検出する問題を局所的トリガ数え上げ問題として定義し，その分散アルゴリズムについて考察する．特に，近傍として距離 ρ 以内にあるプロセッサの集合（ρ-近傍とよぶ）を対象とする ρ-近傍トリガ数え上げ問題について考察する．本報告では (1) 局所分散制御に基づく解法，(2) 集中型制御に基づく解法，(3) クラスタ内制御に基づく解法（(1)，(2) の折衷手法）の 3 つの分散アルゴリズムを提案し，それぞれの通信計算量について考察する．

1　はじめに

1.1　研究背景

ネットワークで接続された n 個の**プロセッサ**から構成される分散システムを用いた分散モニタリングについて考える．分散モニタリングとは，プロセッサに付随したセンサを利用し，交通量や降水量などについての環境モニタリングを実現するシステムである．分散モニタリングにおいて，”交通量”や”降水量”，あるいは，データネットワークにおける”通信量”や”ログイン中のユーザ数”などが，ある一定数を超えたことを検出する問題は，トリガ数え上げ問題とよばれ，これまでにいくつかの研究成果が示されている [1, 2, 3, 4, 5]．トリガ数え上げ問題は，任意のタイミングで任意のプロセッサでトリガが発生しうる状況で，発生したトリガの総数が，ある与えられた値 w（**検出トリガ数**とよぶ）に達したことをいずれかのプロセッサが検出する問題として定式化されている．

トリガ数え上げ問題について，文献 [2] では，総メッセージ数が $O(n \log w)$，1 プロセッサの最大受信メッセージ数が $O(n \log w)$ となる集中型アルゴリズムと，総メッセージ数が $O(n \log n \log w)$，最大受信メッセージ数が $O(n \log n \log w)$ となる木構造に基づく分散アルゴリズムを提案している．

これらの分散アルゴリズムでは，あるプロ

セッサがシステム全体を連携させる代表としての役割を果たし，その他のプロセッサは代表プロセッサに従うという集中型制御に基づいている．一方，分散システムとして，センサネットワークを想定する場合，各プロセッサは限られた電力で動作すると想定することが多く，メッセージ数を減らし，各プロセッサのライフタイムを延ばすための工夫が必要とされている [3, 4, 5]．例えば，文献 [3] では，ネットワークにレイヤと呼ばれる構造を導入することで，総メッセージ数が $O(n \log n \log w)$，最大受信メッセージ数が高い確率で $O(\log n \log w)$ となる乱択アルゴリズムを実現している．

1.2 本報告の目的と結果

トリガ数え上げ問題では，分散システム全体で発生したトリガの総数が一定数を超えたことを検出することを目的とする．しかし，交通量や降水量を対象とする場合，各プロセッサは自身の近傍で発生したトリガを数え上げたいことがある．そこで本報告では，この問題を**局所的トリガ数え上げ問題**として定義し，特に，あるプロセッサから距離 ρ 以内にあるプロセッサの集合（ρ-近傍とよぶ）で発生したトリガの総数が検出トリガ数に達したことを検出する **ρ-近傍トリガ数え上げ問題**について検討する．

本報告では，まず 2 章で分散システムと問題の定義を行い，3 章で (1) 局所分散制御に基づく解法を示す．これは，各プロセッサが ρ-近傍で発生したトリガを集めることで検出トリガに達したことを検出するアルゴリズム SIMPLE_COUNTING による解法である．この解法の通信計算量は，システム全体で発生するトリガ数を W，最大 ρ-次数を $\Delta^{\rho}(G)$ とすると，総メッセージ数 $O(W \cdot \Delta(G)^{\rho})$，最大受信メッセージ数 $O(w)$ となる．これらの計算量はネットワークのトポロジに依存するが，ある ρ-近傍に全てのプロセッサが含まれる（最大 ρ-次数がプロセッサ数 n となる）とき，総メッセージ数は最悪となり，$O(n \cdot W)$ となる．

次に，4 章で (2) 集中型制御に基づく解法を示す．これは，分散システムの中心のプロセッサの 1 つを**サーバ**として指定し，このサーバがシステム全体のトリガ発生状況を管理することで，あるプロセッサの ρ-近傍で発生したトリガの数が検出トリガ数に達したことを検出するアルゴリズム $SERVER\_COUNTING$ による解法である．この解法の通信計算量は，分散システムの半径を $Rad(G)$ とすると，総メッセージ数 $O(W \cdot Rad(G))$，最大受信メッセージ数 $O(W)$ となる．また分散システムのトポロジが，半径 Rad(G) が最大となるようなトポロジのとき，総メッセージ数は最悪となり，$O(n \cdot W)$ となる．

これらの解法 (1),(2) の最悪時の総メッセージ数は，いずれも $O(n \cdot W)$ となるが，既に述べたように，そのトポロジは異なる．そこで 5 章では，解法 (1),(2) の折衷手法として (3) クラスタ内制御に基づく解法を考え，最悪時の通信計算量の改善を図る．この解法では，分散システムにクラスタ構造 (クラスタの集合 $\mathcal{T}$ が分散システムの被覆となっている) を導入し、そのクラスタごとにアルゴリズム $SERVER\_COUNTING$ を適応することで検出トリガに達したことを検出する．このアルゴリズムを $CLUSTER\_COUNTING$ と

よぶ．この解法の通信計算量は，クラスタの最大半径を $Rad(\mathcal{T})$，最大クラスタ次数を $\Delta(\mathcal{T})$ とすると，総メッセージ数 $O(\Delta(\mathcal{T})\cdot Rad(\mathcal{T})\cdot W)$，最大受信メッセージ数 $O(W)$ となる．

	最悪時総メッセージ数
解法 (1)	$O(n\cdot W)$
解法 (2)	$O(n\cdot W)$
解法 (3)	$O(\kappa\cdot n^{1/\kappa}\cdot\rho\cdot W)$

表 1: 提案アルゴリズムの通信計算量

解法 (3) は被覆に関するパラメタにより計算量が異なるため，最大クラスタ次数と平均クラスタ次数の低くなる 2 種類の被覆を与え評価する．最悪時の総メッセージ数は表 1 のように $O(\kappa\cdot n^{1/\kappa}\cdot\rho\cdot W)$ なり，自由に設定可能なパラメタ κ に適当な値を決めることで解法 (1),(2) と比べ改善が可能である．

そして最後に，6 章で本報告の内容をまとめる．

2 諸定義

本章では，ネットワークシステムのモデルについて述べる．次に，ρ-近傍トリガ数え上げ問題を定義し，この問題に対する分散アルゴリズムの評価尺度について説明する．

2.1 システムモデル

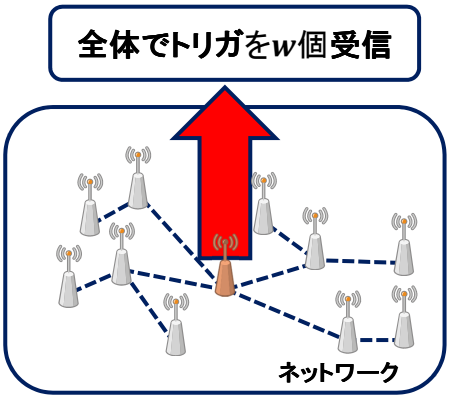
分散システム $G=(V,E)$ は n 個のプロセッサ $V=\{p_1,p_2,\cdots,p_n\}$ と，異なる 2 つのプロセッサ間の通信リンクの集合 E で構成される．各プロセッサ $p_i(1\le i\le n)$ は固有の識別子 $id(p_i)$ を有する．以降では，この $id(p_i)$ を p_i と区別せずに用いる．また，プロセッサの故障は考えないものとする．通信リンク (p_i,p_j) で接続されたプロセッサ p_i,p_j 間ではメッセージの送受信が可能であり，送信したメッセージは破損することなく有限時間内に受信プロセッサに届く．ただし，この遅延に上限がない非同期システムを想定し，リンクは長さに制限のない FIFO キューとしてモデル化する．

2.2 ρ-近傍トリガ数え上げ問題

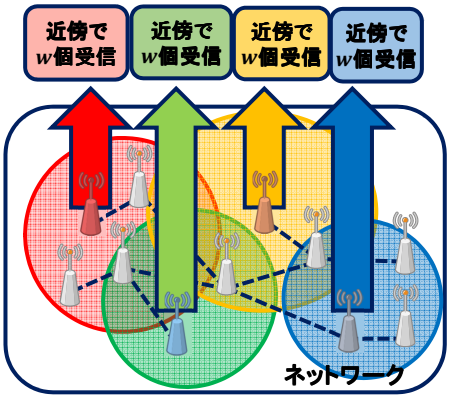
各プロセッサ $p_i(1\le i\le n)$ では，外界から任意のタイミングで**トリガ**が発生するものとする．ただし，トリガがどのプロセッサでどのようなタイミングで発生するかは，前もって決まっていない．また，同一プロセッサで複数のトリガが発生することもある．

このとき，分散システム全体で発生したトリガの総数がある閾値 w（以降，w を**検出トリガ数**とよぶ）に達したことを検出する問題が**大域的トリガ数え上げ問題** [2] である（図

1(a)). ただし w はあらかじめ指定された値とする．これに対し，本報告では分散システムの各プロセッサが自身の近傍でトリガが w 個発生したことを検出する問題を**局所的トリガ数え上げ問題**として新たに定義する（図 1(b)）．



(a) 大域的トリガ数え上げ問題



(b) 局所的トリガ数え上げ問題

図 1: 2 種類のトリガ数え上げ問題

本報告では，局所的トリガ数え上げ問題として，特に，各プロセッサ p_i に対し，距離 ρ 以内にあるプロセッサから構成される **ρ-近傍**（図 2）を考え，この ρ-近傍での発生トリガ数が検出トリガ数 w に達したことを p_i が検出する **ρ-近傍トリガ数え上げ問題**について考察し，そのアルゴリズムの提案を行う．

以下では，各プロセッサ $p_i(1 \leq i \leq n)$ について，その ρ-近傍を $S_{p_i}^{\rho} = \{p \in V | dist(p_i, p) \leq \rho\}$ と表し，$|S_{p_i}^{\rho}|$ を p_i の **ρ-次数**とよぶ．ここで，$dist(p_i, p)$ はプロセッサ p_i, p 間の距離（最短 p_i-p 経路の長さ（辺の数））を表し，$p_i \in S_{p_i}^{\rho}$ であることを注意しておく．また，$S_{p_i}^{\rho}$ は S_i^{ρ} と略記することもある．

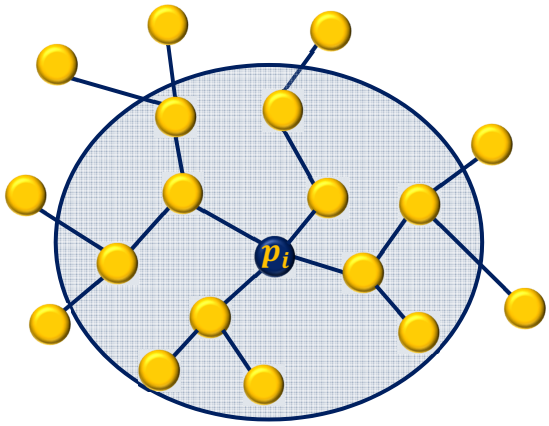


図 2: p_iの 2-近傍

2.3 アルゴリズムの評価尺度

メッセージの送受信には電力が消費される．しかし，センサネットワークなどでは各プロセッサの消費可能電力が限られていることが多いため，送受信するメッセージ数が多いと，ノードの寿命を縮めることになる．そこでメッセージ数に関する次の 2 つの評価尺度を導入し，アルゴリズムの評価を行う．

- **総メッセージ数**：
  分散システム全体で交換されるメッセージの総数．
- **最大受信メッセージ数**：
  システム中の各プロセッサが受信するメッセージ数の最大数．

本報告でのアルゴリズムは，各プロセッサが，トリガの受信時もしくはメッセージの受信時に限られるイベントによりメッセージを送信するイベント駆動型のアルゴリズムである．そのため，メッセージの送信数に関する評価尺度は導入していない．

3 ρ-近傍トリガ数え上げに対する素朴な局所分散型解法

本章では，ρ-近傍トリガ数え上げ問題を解く分散アルゴリズム SIMPLE_COUNTING を示し，その評価を行う．

3.1 アルゴリズム SIMPLE_COUNTING

本アルゴリズムでは，各プロセッサ $p_i(1 \leq i \leq n)$ はトリガが発生すると，このことを $p_i \in S_j^\rho$ を満たす全てのプロセッサ p_j に通知する．各プロセッサは，この通知により自身の ρ-近傍で受信されたトリガの数を検知し，それが w に達したときそのことを検出できる．

このとき，$p_i \in S_j^\rho$ を満たす p_j の集合は S_i^ρ に一致する（つまり，$S_i^\rho = \{p_j \in V | p_i \in S_j^\rho\}$）．よって，$p_i$ でトリガが発生したことの通知は，S_i^ρ の各プロセッサに対して行えばよい．そこで本アルゴリズムでは，各プロセッサ p_i に対し，その ρ-近傍 $S_i^\rho(1 \leq i \leq n)$ の p_i を根とする **幅優先全域木 T_i^ρ** があらかじめ構成されているものとする．このため，各プロセッサ p_i は $|S_i^\rho|$ 個の全域木に含まれることになるが，p_i はこれら $|S_i^\rho|$ 個の全域木の情報（子プロセッサの集合）を保持しているものとする．各プロセッサ p_i は，トリガが発生すると，この全域木 T_i^ρ を用いて S_i^ρ の各プロセッサにトリガの発生を通知する．以降では，各プロセッサの動作について説明を行う．

各プロセッサ p_i は，ρ-近傍 S_i^ρ で発生したトリガ数を記録する **カウンタ $C(p_i)$**（初期値は 0）を保持している．p_i でトリガが発生した場合，まず自身のカウンタ $C(p_i)$ をインクリメントする．そして p_i の ρ-近傍 S_i^ρ に属するすべ

てのプロセッサに，全域木 T_i^ρ を用いて，トリガの発生を通知する．これは T_i^ρ を用いて，S_i^ρ の全プロセッサにメッセージ $\boldsymbol{TRIGGER_{p_i}}$ を配信することで行う（図 3）．

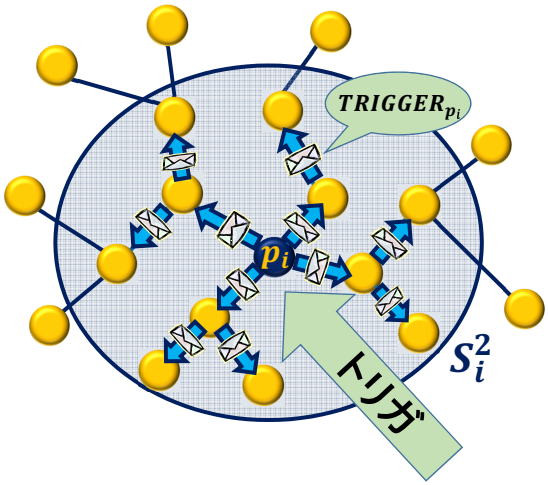


図 3: プロセッサ p_i のトリガ発生時の動作（$\rho = 2$）

このメッセージ $TRIGGER_{p_i}$ を受信したプロセッサ p_j は，自分のカウンタ $C(p_j)$ をインクリメントし，全域木 T_i^ρ に従ってメッセージを転送する．そしてカウンタの値が w に達したプロセッサは，このことを検出してアルゴリズムを終了する．

最後に，本手法の疑似コードを Algorithm1 に示す．

3.2 アルゴリズムの評価

まず総メッセージ数についての評価を行う．このアルゴリズムでは，トリガが発生したプロセッサ p_i は，自身の ρ-近傍 S_i^ρ に属する全てのプロセッサにメッセージ $TRIGGER_{p_i}$ を配信する．これは，S_i^ρ の全域木 T_i^ρ を用いて行うので，これに要するメッセージ数は $|S_i^\rho| - 1$ である．よって，$\Delta(G)^\rho = max\{\ |S_i^\rho|\ |p_i \in V\}$ を分散システム G の最大 ρ-次数，W を分散システム全体で発生したトリガ数とすれば，総メッセージ数は $O(W \cdot \Delta(G)^\rho)$ となる．

次に，最大受信メッセージ数について評価を行う．プロセッサ p_i が受信するメッセージは全て p_iのρ-近傍 S_i^ρ で発生したトリガである．従って，w 個のトリガが発生した時点でアルゴリズムは停止するので，最大受信メッセージ数は高々w である．これより，以下の定理が導かれる．

定理 1

最大 ρ-次数が $\Delta(G)^\rho$ の分散システム G において，分散アルゴリズム SIMPLE_COUNTING は総メッセージ数 $O(W \cdot \Delta(G)^\rho)$，最大受信メッセージ数 $O(w)$ で ρ-近傍トリガ数え上げ問題を解く．ただし，W はシステム全体で発生したトリガ数，w は検出トリガ数とする．

4 ρ-近傍トリガ数え上げに対する集中型解法

本章では，ρ-近傍トリガ数え上げ問題を解く分散アルゴリズム SERVER_COUNTING を示し，その評価を行う．

4.1 アルゴリズム SERVER_COUNTING

まず，分散システム G の**半径**と**中心**を定義する．

定義 1 分散システム $G = (V, E)$ におけるプロセッサ $v \in V$ の離心数 $Ecc(v, G)$ を以下のように定義する．

$$Ecc(v, G) = \max_{w \in V}\{dist_G(v, w)\}$$

この離心率 $Ecc(v, G)$ が最も小さくなるようなプロセッサ u を分散システム G の中心と呼び，$Ecc(u, G)$ を分散システム G の半径 $Rad(G)$ とよぶ．

$$Rad(G) = \min_{v \in V}\{Ecc(v, G)\}$$

この中心のプロセッサの 1 つを**サーバ s** として指定する．サーバ s は分散システム G のトポロジを知っており，発生したトリガの数を ρ-近傍 $S_i^\rho (1 \leq i \leq n)$ ごとに管理している．これらの情報から，サーバ s は ρ-近傍 S_i^ρ で発生したトリガの数が w に達したことを検知したとき，プロセッサ p_i に通知する．このようにして，ρ-近傍トリガ数え上げ問題を解く，というのがアルゴリズム SERVER_COUNTING の概要である．以降，アルゴリズムについて詳細に述べていく．

まず分散システム G に対し，サーバ s を根とする幅優先全域木 T_G があらかじめ構成されている．各プロセッサ p_i は，この全域木 T_G の情報（親プロセッサ，子プロセッサの集合）を保持している．サーバ s は，これに加え，分散システム G のトポロジ情報（プロセッサの識別子を含む）も保持する．また，3 章では各プロセッサがそれぞれカウンタを保持していたが，本章ではサーバ s が各 ρ-近傍クラスタ S_i に対し，カウンタ $C(p_i)$ を保持している．

次に，各プロセッサの動作について説明する．本アルゴリズムでは，サーバ s とそれ以外のプロセッサ p_i では動作が異なるため，それぞれ説明を行う．

- サーバ s 以外のプロセッサ p_i の動作（図 4）
  - トリガが発生した場合
    トリガが発生したというメッセージをサーバ s に全域木 T_G を用いて送信する．このメッセージには，サーバ s が送り主が分かるように p_i の識別子をメッセージに含める．以降，このメッセージを $\boldsymbol{TRIGGER(i)}$ とよぶ．
  - メッセージを受信した場合
    p_i が受信するメッセージは，トリガが発生した他のプロセッサ $p_j(1 \leq j \leq n)$ から送信された $TRIGGER(j)$ メッセージである．よってこのメッセージを受信した場合，全域木 T_G における親プロセッサに $TRIGGER(j)$ を転送することで，サーバ s へメッセージを転送する．

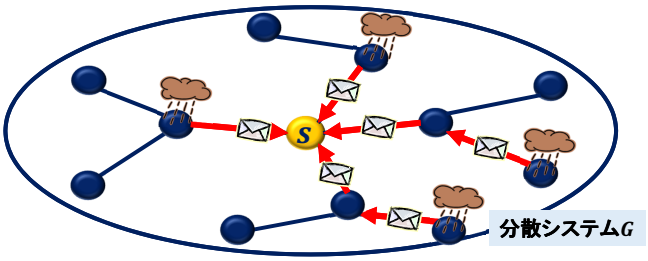


図 4: サーバ s 以外のプロセッサの動作

- サーバ s の動作
  - トリガが発生した場合
    $s \in S_j^\rho(1 \leq j \leq n)$ を満たす ρ-近傍 S_j^ρ のカウンタ $C(p_j)$（s が属する ρ-近傍のカウンタ）をインクリメントする．
  - メッセージ $TRIGGER(j)$ を受信した場合
    $p_j \in S_k^\rho$ を満たす ρ-近傍 S_k^ρ のカウンタ $C(p_k)$（p_j が属する ρ-近傍のカウンタ）をインクリメントする．

以上の処理を繰り返して，ある ρ-近傍 S_j^ρ で w 個のトリガを検出した（$C(p_j) = w$ になった）場合，サーバ s は幅優先全域木 T_G を用いて，メッセージ $\boldsymbol{ALARM(j)}$ を送り，そのことをプロセッサ p_j に知らせてアルゴリズムを終了する．

本アルゴリズムの疑似コードを示す．Algorithm2 がサーバ以外のプロセッサ p_i のアルゴリズム，Algorithm3 がサーバ s のアルゴリズムである．

したトリガ数，w は検出トリガ数とする．

4.2 アルゴリズムの評価

まず総メッセージ数についての評価を行う．このアルゴリズムでは，トリガが発生したプロセッサ p_i は，サーバ s へメッセージ $TRIGGER(i)$ を送信する．サーバは分散システム G の中心のプロセッサなので，これに要するメッセージ数は高々 $Rad(G)$ となる．よって，W を分散システム全体で発生したトリガ数とすれば，総メッセージ数は $O(W \cdot Rad(G))$ となる．この総メッセージ数は，半径 $Rad(G)$ が最大となるようなトポロジのとき（図 5）に最悪となり，総メッセージ数は $O(n \cdot W)$ となる．

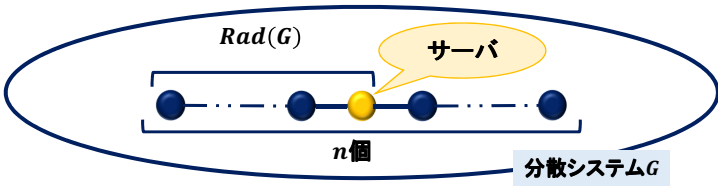


図 5: 総メッセージ数最悪時のトポロジ

次に，最大受信メッセージ数について評価を行う．プロセッサ p_i でトリガが発生したことを通知するメッセージは全てサーバ s に集められる．従って，最大受信メッセージ数はサーバが受信する W 個となる．これより，以下の定理が導かれる．

定理 2 半径が $Rad(G)$ の分散システム G において，分散アルゴリズム SERVER_COUNTING は総メッセージ数 $O(W \cdot Rad(G))$, 最大受信メッセージ数 W で ρ-近傍トリガ数え上げ問題を解く．ただし，W はシステム全体で発生

5 ρ-近傍トリガ数え上げに対するクラスタ型解法

本章ではまず，分散システム G の誘導部分システムであるクラスタとそのクラスタを粗大化した被覆を定義し，この被覆が与えられたときに，ρ-近傍トリガ数え上げ問題を解く分散アルゴリズム CLUSTER_COUNTING を示す．そして，最大次数の低い被覆，平均次数が低い被覆それぞれが与えられた場合の CLUSTER_COUNTING の通信計算量の評価を行う．

5.1 クラスタと被覆

まず**クラスタ**を定義する．分散システム $G = (V, E)$ のプロセッサの部分集合 $C(\subseteq V)$ に対し，両端のプロセッサが C に含まれる通信リンクの集合を $E' = \{(p_i, p_j) \in E | p_i, p_j \in C, i \neq j\}$ とする．このとき，C による誘導部分システム $G(C) = (C, E')$ をクラスタとよぶ．

特に混乱が生じない限り，クラスタ $G(C)$ を C と略記する．また，クラスタ $G(C)$ とプロセッサの集合を区別せずに扱う．次に，クラスタ C の**半径**と**中心**を定義する．

定義 2 クラスタ C におけるプロセッサ $v \in C$ の離心数 $Ecc(v, C)$ を以下のように定義する．

$$Ecc(v, C) = \max_{w \in C}\{dist_C(v, w)\}$$

この離心率 $Ecc(v, C)$ が最小となるプロセッサ u をクラスタ C の中心と呼び，$Ecc(u, C)$ をクラスタ C の半径 $Rad(C)$ とよぶ．

$$Rad(C) = \min_{v \in C}\{Ecc(v, C)\}$$

クラスタの集合 $\mathcal{C} = \{C_1, ..., C_m\}$ が，分散システム G のすべてのプロセッサを含む（$\cup\mathcal{C} = V$）とき，クラスタの集合 $\mathcal{C}$ を**被覆**とよぶ．また，各クラスタ $\{C_1, ..., C_m\}$ の半径のうち最大のものを被覆 $\mathcal{C}$ の **半径** $\boldsymbol{Rad(\mathcal{C})}$，各プロセッサ $p_i (1 \leq i \leq n)$ が属するクラスタの数を p_i の **クラスタ次数** $\boldsymbol{c\text{-}deg(p_i)} (= |\{C \in \mathcal{C} | p_i \in C\}|)$ とよぶ．また，以下のように，被覆 $\mathcal{C}$ の **最大クラスタ次数** $\boldsymbol{\Delta(\mathcal{C})}$ と **平均クラスタ次数** $\boldsymbol{\bar{\Delta}(\mathcal{C})}$ を定義する．

定義 3 最大クラスタ次数 $\Delta(\mathcal{C})$ を以下のように定義する．

$$\Delta(\mathcal{C}) = \max_{v \in V}\{c\text{-}deg(v)\}$$

定義 4 平均クラスタ次数 $\bar{\Delta}(\mathcal{C})$ を以下のように定義する．

$$\bar{\Delta}(\mathcal{C}) = \frac{1}{n} \cdot \sum_{v \in V} c\text{-}deg(v)$$

また本章では，プロセッサ p_i の ρ-近傍による誘導部分システムを p_i の ρ-近傍クラスタ S_i^ρ，すべてのプロセッサの ρ-近傍クラスタの集合 $\mathcal{S} = \{S_1, ..., S_n\}$ を ρ-近傍被覆 とよぶことにする．

5.2 アルゴリズム CLUSTER_COUNTING

本節では，被覆を利用して ρ-近傍トリガ数え上げ問題を解く分散アルゴリズム CLUSTER_COUNTING を示す．

本報告で利用する被覆 $\mathcal{T}$ は，いくつかの ρ-近傍クラスタを併合（**粗大化**，図 6）して得られるクラスタで構成され，$\mathcal{T} = \{T_1, T_2, ..., T_m\} (m \leq n)$ と表す．つまり，$V = \bigcup_{j=1..m} T_j$ であり，

各 $S_i^\rho(1 \leq j \leq n)$ に対し $S_i \subseteq T_j$ なるクラスタ $T_i(1 \leq j \leq m)$ が存在する．

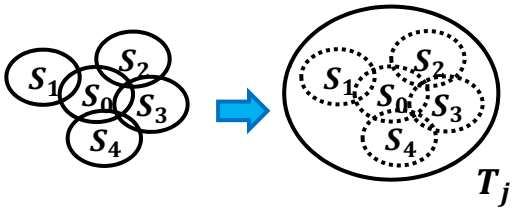


図 6: クラスタの粗大化

$\mathcal{T}$ の各クラスタ $T_j(1 \leq j \leq m)$ において，中心のプロセッサの 1 つがサーバとして指定されているものとし，このプロセッサを s_j と表す．サーバ s_j はクラスタ T_j のトポロジを知っており，T_j に属するプロセッサで発生したトリガの数を T_j に属するクラスタ S_i^ρ ごとに管理している．これらの情報から，サーバ s_j は $S_i^\rho \subseteq T_j$ なる ρ-近傍クラスタ S_i^ρ で発生したトリガの数が w に達したとき，プロセッサ p_i に通知する．つまり，各クラスタ T_j でアルゴリズム $SERVER\_COUNTING$ を実行することにより ρ-近傍トリガ数え上げ問題を解く，というのがアルゴリズム CLUSTER_COUNTING の概要である．以降，アルゴリズムについて詳細に述べていく．

まず各クラスタ $T_j(1 \leq j \leq m)$ に対し，サーバ s_j を根とする幅優先全域木 U_j があらかじめ構成されているものとする．各プロセッサ p_i は，高々最大クラスタ次数 $\Delta(\mathcal{T})$ 個の全域木に含まれることになるが，p_i はこれらの全域木の情報（親プロセッサ，子プロセッサの集合）を保持しているものとする．サーバ s_j は，各プロセッサが持つ情報に加え，クラスタ T_j のトポロジ情報（プロセッサの識別子を含む）を把握している．また，3 章では各プロセッサがそれぞれカウンタを保持していたが，本章ではサーバ s_j が T_j に含まれる各 ρ-近傍クラスタ S_i^ρ に対し，カウンタ $C(p_i)$ を保持している．

次に，各プロセッサ p_i の動作について説明する．本アルゴリズムでは，サーバとそれ以外のプロセッサでは動作が異なるため，それぞれ説明を行う．

- サーバ以外のプロセッサ p_i の動作
  - トリガが発生した場合
    トリガが発生したというメッセージを，$p_i \in T_j$ を満たす全てのクラスタ T_j のサーバ s_j に全域木 U_j を用いて送信する．このメッセージには，送り主と宛先が分かるように，送り主 p_i と宛先 s_j の識別子をメッセージに含める．以降，このメッセージを $\boldsymbol{TRIGGER_{p_i\text{-}s_j}}$ とよぶ．
  - メッセージを受信した場合
    p_i が受信するメッセージは，トリガが発生した他のプロセッサ p_l からサーバへ送信された $TRIGGER_{p_l\text{-}s_j}$ メッセージである．よってこのメッセージを受信した場合，全域木 U_j に従いサーバ s_j へ向けメッセージを転送する．
- p_i がサーバ s_k のときの動作
  - トリガが発生した場合
    $s_k \in S_l^\rho$ を満たす p_l のカウンタ $C(p_l)$（s_k が属する ρ-近傍クラスタのカウンタ）をインクリメント

する．ただしサーバ s_k が他のクラスタ $T_{s'}$ にも所属している場合は，$T_{s'}$ のサーバにトリガの発生を伝えるための，上記のサーバ以外のプロセッサの動作を実行する．

– メッセージを受信した場合
自分宛ではない $TRIGGER_{p_l\text{-}s_j}$ メッセージ（$s_j \neq s_k$）を受信した場合，上記のサーバ以外のプロセッサの動作に従いメッセージをサーバ S_j に向けて転送する．自分宛の $TRIGGER_{p_l\text{-}s_k}$ メッセージを受信した場合，$p_l \in S_t^\rho$ を満たす p_t のカウンタ $C(p_t)$（p_l が属する ρ-近傍クラスタのカウンタ）をインクリメントする．

以上の処理を繰り返して，ある ρ-近傍 S_l で w 個のトリガを検出した場合（あるカウンタ $C(p_l)$ の値が w になった場合），サーバ s_k は，T_k の幅優先全域木 U_k を用いて，このことをプロセッサ p_l に知らせる．このメッセージを $\boldsymbol{ALARM_{s_k}(L)}$ とよぶ．ただし，L は w 個のトリガを検出したプロセッサの識別子の集合である．

本アルゴリズムの疑似コードを示す．Algorithm4 がサーバ以外のプロセッサ p_i のアルゴリズム，Algorithm5 がプロセッサ p_i がサーバ s_k のときのアルゴリズムである．

5.3 アルゴリズムの評価

まず総メッセージ数についての評価を行う．このアルゴリズムでは，トリガが発生したプロセッサ p_i は，$p_i \in T_j$ を満たすサーバ s_j 全てに幅優先全域木 U_j を用いてメッセージ $TRIGGER_{p_i\text{-}s_j}$ を送信する．p_i から s_j まで $TRIGGER_{p_i\text{-}s_j}$ を届けるのに高々 $Rad(\mathcal{T})$ 個のメッセージが必要である．また p_i は，高々 $\Delta(\mathcal{T})$ のクラスタに属するので，トリガが1つ発生した際に必要なメッセージは高々 $\Delta(\mathcal{T}) \cdot Rad(\mathcal{T})$ となる．よって，システム全体で発生するトリガ数を W とすると，メッセージ数は $O(\Delta(\mathcal{T}) \cdot Rad(\mathcal{T}) \cdot W)$ となる．

定理 3 システム全体で発生するトリガ数を W とするとき，アルゴリズム $CLUSTER\_COUNTING$ の総メッセージ数は以下のようになる．

$$O(\Delta(\mathcal{T}) \cdot Rad(\mathcal{T}) \cdot W)$$

次に，最大受信メッセージ数について評価を行う．各プロセッサ p_i が受信するメッセージには2種類ある．1つは $ALARM_{s_k}(l)$ メッセージである．これは p_i が受信するのはアルゴリズムの終了時の一度のみである．もう1つは，トリガの発生を知らせる $TRIGGER_{p_k\text{-}s_j}$ である．$TRIGGER_{p_k\text{-}s_j}$ メッセージは，各プロセッサでトリガが1つ発生するたびに1つ生成される．システム全体で発生するトリガ数を W とすると，すべての $TRIGGER_{p_k\text{-}s_j}$ がプロセッサ p_i を経由する場合に最大受信メッセージ数が最大となる．よって，$ALARM_{s_k}(l)$ メッセージと合わせ，最大受信メッセージ数は高々 $W + \Delta(\mathcal{T})$ となる．

定理 4 システム全体で発生するトリガ数を W とするとき，アルゴリズム CLUSTER_COUNTING の最大受信メッセージ数は $O(W + \Delta(\mathcal{T}))$ となる．

5.3.1 最大クラスタ次数の低い被覆を与えた場合

本節では，5.2 節のアルゴリズム CLUSTER_COUNTING に最大クラスタ次数が低い被覆を与えた場合についての評価を行う．

最大クラスタ次数が低い被覆を構成するアルゴリズムには，MAX_COVER というアルゴリズムがある [6]．これは，ある被覆 $\mathcal{C}$ を入力として，C に属するクラスタを粗大化することで最大クラスタ次数の低い被覆 $\mathcal{T}$ を出力するアルゴリズムである．MAX_COVER が出力する被覆 $\mathcal{T}$ について，以下の補題 1 が成り立つことが知られている．

補題 1 [6] パラメータを κ，入力を被覆 $\mathcal{C}$ とした時，アルゴリズム MAX_COVER は以下の性質を満たす被覆 $\mathcal{T}$ を出力する．ただし，$|\mathcal{C}|$ は被覆 $\mathcal{C}$ のクラスタの数とする．

- $Rad(\mathcal{T}) \leq (2\kappa - 1)Rad(\mathcal{C})$
- $\Delta(\mathcal{T}) \leq 2\kappa \cdot |\mathcal{C}|^{\frac{1}{\kappa}}$

MAX_COVER に ρ-近傍被覆 $\mathcal{S}$ を入力すれば，$\mathcal{S}$ のクラスタの数は n 個であることより，次のような最大クラスタ次数の低い被覆 $\mathcal{T}$ が得られる．

- $Rad(\mathcal{T}) \leq (2\kappa - 1)\rho$
- $\Delta(\mathcal{T}) \leq 2\kappa \cdot n^{\frac{1}{\kappa}}$

この被覆 $\mathcal{T}$ を用いると，アルゴリズム CLUSTER_COUNTING の通信計算量は定理 3，4，補題 1 より次のようになる．

定理 5 最大クラスタ次数の低い被覆 $\mathcal{T}$ を用いると，アルゴリズム CLUSTER_COUNTING の総メッセージ数は，$O(\Delta(\mathcal{T}) \cdot Rad(\mathcal{T}) \cdot W) = O(\kappa^2 \cdot n^{\frac{1}{\kappa}} \cdot \rho \cdot W)$ となる．また，最大受信メッセージ数は $O(W + \kappa \cdot n^{1/\kappa})$ となる．

5.3.2 平均クラスタ次数の低い被覆を与えた場合

本節では，5.2 節のアルゴリズム CLUSTER_COUNTING に平均クラスタ次数が低い被覆を与えた場合についての評価を行う．

平均クラスタ次数が低い被覆を構成するアルゴリズムには，AV_COVER というアルゴリズムがある [6]．これは，ある被覆 $\mathcal{C}$ を入力として，C に属するクラスタを粗大化することで平均クラスタ次数の低い被覆 $\mathcal{T}$ を出力するアルゴリズムである．AV_COVER が出力する被覆 $\mathcal{T}$ について，以下の補題 2 が成り立つことが知られている．

補題 2 [6] パラメータを κ，入力を被覆 $\mathcal{C}$ とした時，アルゴリズム AV_COVER は以下の性質を満たす被覆 $\mathcal{T}$ を出力する．

- $Rad(\mathcal{T}) \leq (2\kappa + 1)Rad(\mathcal{C})$
- $\overline{\Delta}(\mathcal{T}) \leq n^{1/\kappa}$

AV_COVER に ρ-近傍被覆 $\mathcal{S}$ を入力すれば，次のような平均クラスタ次数の低い被覆 $\mathcal{T}$ が得られる．

- $Rad(\mathcal{T}) \leq (2\kappa+1)\rho$
- $\overline{\Delta}(\mathcal{T}) \leq n^{1/\kappa}$

この被覆 $\mathcal{T}$ を用いると，アルゴリズム CLUSTER_COUNTING の通信計算量は定理 3，4，補題 2 より次のようになる．

定理 6 平均クラスタ次数の低い被覆 $\mathcal{T}$ を用いると，アルゴリズム CLUSTER_COUNTING の総メッセージ数は，定理 3 より $O(\Delta(\mathcal{T})\cdot Rad(\mathcal{T})\cdot W) = O(\kappa\cdot n^{1/\kappa}\cdot\rho\cdot W)$ となる．また，最大受信メッセージ数は，定理 4 より $O(W+\kappa\cdot n^{1/\kappa})$ となる．

6 まとめ

本報告では，ρ-近傍トリガ数え上げ問題を提起し，この問題を解く分散アルゴリズムを 3 つ示した．最初に，各プロセッサがそれぞれ ρ-近傍での発生トリガ数を数えることで検出するアルゴリズム SIMPLE_COUNTING を示した．このアルゴリズムは，疎なトポロジにおいてメッセージ数の効率が良い．次に，ある 1 つのプロセッサが，すべての ρ-近傍での発生トリガ数を数えることで検出するアルゴリズム SERVER_COUNTING を示した．このアルゴリズムは，SIMPLE_COUNTING と対照的に，密なトポロジにおいてメッセージ数の効率が良い．最後に，これらのアルゴリズムの折衷手法として，クラスタごとに SERVER_COUNTING を適応する CLUSTER_COUNTING を示した．このアルゴリズムでは，最悪時の総メッセージ数に関して，先の 2 つより改善されることを示した．

本報告で提案した分散アルゴリズムはいずれも，トリガが発生するたびにトリガの発生を通知している．従って，トリガが非負の実数の重みを持ち，その和が閾値に達したことを検出する**局所的重み付きトリガ数え上げ問題**に対しても，重みを通知することにより，これらの分散アルゴリズムを適用できる．さらに，各プロセッサで数え上げるトリガの重みが，トリガの重みとトリガが発生したプロセッサからの距離に依存する場合にも，重みと距離を考慮することにより，これらの分散アルゴリズムを適用可能である．ただし，アルゴリズム SIMPLE_COUNTING では，トリガの重みに加えて距離の通知が必要となる．

局所的重み付きトリガ数え上げ問題で，トリガの負の重みを許すことも可能である．この場合にもこれらのアルゴリズムを適用できるが，非同期式システムでは，重み付きトリガ数え上げ問題を正確に解くことは不可能である．これは，重みが非負の場合は，発生したトリガの重みの和が閾値以上であるという性質は安定（一度でも満たされると，それ以降も満たされる）であるが，負の重みを許すと安定ではなくなるためである．このため，非同期式システムでは，誤検出（フォールスポジティブ）か検出漏れ（フォールスネガティブ）の発生を避けることはできない．

文献 [2] の木構造に基づくアルゴリズムでは，トリガが発生するたびにそのことの通知を避けることにより,（大域的）トリガ数え上げアルゴリズムのメッセージ数を削減している．この手法は，ネットワーク全体でのトリガ数の数え上げには有効であるが，局所的トリガ数え上げへの適用は容易でない．局所的トリガ数え上げ問題アルゴリズムのメッセージ数の改善は今後の課題である．

参考文献

[1] Ram Keralapura, Graham Cormode, and Jeyashankher Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pp. 289–300, 2006.

[2] Rahul Garg, Vijay K.Garg, and Yogish Sabharwal. Scalable algorithms for global snapshots in distributed systems. In *Proceedings of the 20th annual international conference on Supercomputing(ICS)*, pp. 269–277, 2006.

[3] Venkatesan T.Chakaravarthy, Anamitra R.Choudhury, Vijay K.Garg, and Yogish Sabharwal. An efficient decentaralized algorithm for the distributed trigger counting problem. In *12th International Conference on Distributed Computing and Networking(ICDCN)*, pp. 53–64, 2011.

[4] Sushanta Karmakar, A. Chandrakanth Reddy, and Yogish Sabharwal. Improved algorithm for the distributed trigger counting problem. *IEEE International Parallel and Distributed Processing Symposium(IPDPS)*, pp. 515–523, 2011.

[5] Venkatesan T.Chakaravarthy, Anamitra R.Choudhury, and Yogish Sabharwal.

An improved algorithm for distributed trigger counting in ring. *The Computer Journal*, 2013.

[6] David Peleg. *Distributed Computing: a locality-sensitive approach.* Society for Industrial and Applied Mathematics, 1987.

Algorithm 1 SIMPLE_COUNTING（p_i の動作）

const

w：検出トリガ数

p_j：任意のプロセッサ $(1 \leq j \leq n)$

$Child_j(p_i)$：全域木 T_j^{ρ} における p_i の子プロセッサの集合

var

$C(p_i) \leftarrow 0$：各カウンタを初期化

function

1: **while** 各カウンタ $C(p_i) \neq w$
2: **begin**
3: **if**（トリガが発生） **then**
4: $C(p_i) \leftarrow C(p_i) + 1$：カウンタをインクリメント
5: T_i^{ρ}における p_iの子 $Child_i(p_i)$ へ $TRIGGER_{p_i}$ メッセージを送信
6: **else if**（$TRIGGER_{p_j}$ メッセージを受信）**then**
7: $C(p_i) \leftarrow C(p_i) + 1$：カウンタをインクリメント
8: T_j^{ρ}における p_iの子 $Child_j(p_i)$ へ $TRIGGER_{p_j}$ メッセージを転送
9: **end**
10: w 個のトリガを検出しアルゴリズムを終了する

Algorithm 2 SERVER_COUNTING（サーバ以外のプロセッサ $p_i(1 \leq i \leq n)$ の動作）

const

w：検出トリガ数

$parent(p_i)$：全域木 T_G における p_i の親プロセッサ

$Child(p_i)$：全域木 T_G における p_i の子プロセッサの集合

L：w 個のトリガを検出したプロセッサの識別子の集合

function

1: **while** $ALARM(L)$ が未受信
2: **begin**
3: **if** （トリガが発生） **then**
4: p_i の親 $parent(i)$ へ $TRIGGER(p_i)$ メッセージを送信
5: **else if** （$TRIGGER(j)$ メッセージを受信）**then**
6: p_i の親 $parent(i)$ へ $TRIGGER(j)$ メッセージを転送
7: **end**
8: **if** （$i \in L$）**then**
9: S_i^ρ が検出トリガ数に達したことを検出
10: p_i の子 $Child_j(p_i)$ へ $ALARM(L)$ メッセージを転送
11: アルゴリズム終了

Algorithm 3 SERVER_COUNTING（サーバ s の動作）

const

w：検出トリガ数

$parent(p_i)$：全域木 T_G における p_i の親プロセッサ

$Child(p_i)$：全域木 T_G における p_i の子プロセッサの集合

$L = \{p_l | C(p_l) = w\}$：w 個のトリガを検出したプロセッサの識別子の集合

var

$C(p_i) \leftarrow 0$：各 $p_i (1 \leq i \leq n)$ に対しカウンタを初期化

function

1: **while** （任意の $p_i (1 \leq i \leq n)$ に対し $C(p_i) \neq w$）
2: **begin**
3: **if** （トリガが発生） **then**
4: **for** $\{p_i \mid s \in S_i^{\rho}\}$ /* s を含む ρ-近傍のカウンタを増やす */
5: $C(p_i) \leftarrow C(p_i) + 1$
6: **else if** （$TRIGGER(i)$ メッセージを受信）**then**
7: **for** $\{p_j \mid p_i \in S_j^{\rho}\}$ /* p_i を含む ρ-近傍のカウンタを増やす */
8: $C(p_j) \leftarrow C(p_j) + 1$
9: **end**
10: **for** 各 $p_i (1 \leq i \leq n)$ に対し
11: **if** （$C(p_i) = w$） **then**
12: プロセッサの識別子の集合 L に p_i を追加
13: サーバ s の子 $Child(s)$ へ $ALARM(L)$ メッセージを送信
14: アルゴリズム終了

Algorithm 4 $CLUSTER\_COUNTING$（サーバ以外のプロセッサ $p_i(1 \leq i \leq n)$ の動作）

const

w：検出トリガ数

$Server(p_i) = \{s_j \mid p_i \in T_j\}$：$p_i$ を含むクラスタのサーバの集合

$parent_j(p_i)$：全域木 U_j における p_i の親プロセッサ $(s_j \in Server(p_i))$

$Child_j(p_i)$：全域木 U_j における p_i の子プロセッサの集合 $(s_j \in Server(p_i))$

L：w 個のトリガを検出したプロセッサの識別子の集合

function

1: **while** どのサーバ $s_j(1 \leq j \leq m)$ からも $ALARM_{s_j}(L)$ が未受信
2: **begin**
3: **if**（トリガが発生）**then**
4: **for** 各 $s_j \in Server(p_i)$ /* s_j ：p_i を含むクラスタのサーバ */
5: U_j における p_i の親 $parent_j(p_i)$ へ $TRIGGER_{p_i-s_j}$ メッセージを送信
6: **else if**（$TRIGGER_{p_k-s_j}$ メッセージを受信）**then**
7: U_j における p_i の親 $parent_j(p_i)$ へ $TRIGGER_{p_k-s_j}$ メッセージを転送
8: **end**
9: **if**（$i \in L$）**then**
10: S_i^{ρ} が検出トリガ数に達したことを検出
11: U_j における p_i の子 $Child_j(p_i)$ へ $ALARM_{s_j}(L)$ メッセージを転送
12: アルゴリズム終了

Algorithm 5 $CLUSTER\_COUNTING$（サーバ s_k の動作）

const

w：検出トリガ数

$Server(s_k) = \{s_j \mid s_k \in T_j\}$：$s_k$ を含むクラスタのサーバの集合

$parent_j(s_k)$：全域木 U_j における s_k の親プロセッサ ($s_j \in Server(s_k)$)

$Child_k$：全域木 U_k における s_k の子プロセッサの集合

$L = \{p_l | C(p_l) = w\}$：w 個のトリガを検出したプロセッサの識別子の集合

var

$S_i^\rho \subseteq T_k$なる各 p_iに対し $C(p_i) \leftarrow 0$：各カウンタを初期化

function

while （$S_i^\rho \subseteq T_k$なる各 p_iに対し $C(p_i) \neq w$）**and**
（どのサーバ s_jからも $ALARM_{s_j}(L)$ が未受信）
begin
  if （トリガが発生） **then**
    for $s_k \in S_i^\rho$ なる 各 p_i に対し　/* s_k を含む ρ-近傍のカウンタを増やす */
      $C(p_i) \leftarrow C(p_i) + 1$
    for 各 $s_j \in Server(s_k)$　/* s_j ：s_k を含むクラスタのサーバ */
      U_jにおける s_k の親 $parent_j(s_k)$ へ $TRIGGER_{s_k - s_j}$ メッセージを送信
  else if （$TRIGGER_{p_i - s_j}$ メッセージを受信）**then**
    if （$s_j = s_k$） **then**　/* 自分宛のメッセージの場合 */
      for $s_j \in S_i^\rho$ なる 各 p_i に対し　/* p_i を含む ρ-近傍のカウンタを増やす */
        $C(p_i) \leftarrow C(p_i) + 1$
    else
      U_jにおける s_k の親 $parent_j(s_k)$ へ $TRIGGER_{p_i - s_j}$ メッセージを転送
end
if （$S_i^\rho \subseteq T_k$ なる 各 p_i に対し $C(p_i) = w$） **then**
  プロセッサの識別子の集合 L に p_i を追加
  $Child_k$ へ $ALARM_{s_k}(L)$ メッセージを送信し p_i に通知
else if （$ALARM_{s_j}(L)$ 受信）**then**
  $S_{s_k}^\rho$ が検出トリガ数に達したことを検出
  U_j における p_i の子 $Child_j(s_k)$ へ $ALARM_{s_j}(L)$ メッセージを転送
アルゴリズム終了