# 故障を考慮した高速LANのモデルと分散アルゴリズムに関する研究

1991年1月
齊藤明紀

# 故障を考慮した高速LANのモデルと分散アルゴリズムに関する研究

1991年1月

齊藤明紀

# 内容梗概

本論文は筆者が大阪大学大学院基礎工学部研究科(物理系専攻)の学生として，都倉研究室において行なった研究のうち，高速LANを想定したモデル上での故障を考慮した分散アルゴリズムに関する研究をまとめたものである．

近年，高速なローカルエリアネットワーク(LAN)の普及により，LANによる分散環境下で複数の計算機が協調して動作するようなソフトウェアに対する要求が高まっている．ネットワークを構成する各計算機に分散されている情報をもとに，互いに通信によって情報を交換することで問題を解くアルゴリズムは分散アルゴリズムと呼ばれる．

これまでに，多くの分散アルゴリズムが提案され，研究されている．分散アルゴリズムで一般的に用いられているネットワークモデルやアルゴリズムの評価尺度は，通信が高価でかつ遅いことを前提にしている．一方，イーサネットなどの高速LANは安価で遅延時間の短い通信を実現している．そのため，既に提案されている分散アルゴリズムで用いられているネットワークモデルでは高速LANの特性を利用したアルゴリズムが構成が難しい．

またネットワークを数日ないし数十日以上に渡って連続して稼働させることは普通に行なわれる．この時，ネットワーク内の個々の計算機に故障などによる一時的な停止や再起動が起こることは一般には避けられず，これに対処するアルゴリズムの研究は重要な課題である．しかし，分散アルゴリズムの研究で広く用いられているモデルである非同期ネットワークはメッセージの遅延に上限を設けないものであり，アルゴリズムの動作中に新たに故障が起こったりあるいは故障からの回復も起こり得るような状況がうまく扱えない．

本論文ではまず高速LANの特性を反映したネットワークモデルを作り，その上で計算機の停止や再起動が起こり得る状況での分散アルゴリズムについて考察している．

本論文の第1章では分散アルゴリズムと本研究での結果について概説している．

第2章では高速LANの性質を考察し，高速LANを想定したネットワークモデルを提案している．このモデルでは計算機を，停止し，また再起動するものとして定義している．またこのモデルでのネットワークの結合性についても定義している．

第3章ではアルゴリズムの動作中にさらに停止や起動が起こり得る状況で情報の更新を繰り返し行う問題を解くアルゴリズムを示し，正当性を証明している．

第4章では完全L分木を維持する問題について考察し，本モデル上の分散アルゴリズムを評価する計算量の尺度について考察している．さらに問題を解くアルゴリズムを示し、そのメッセージ複雑度および局所計算量を評価している。

ネットワーク中の全ノードが参加するサーバクライアント関係を考えた時，サーバへの負荷集中を避けるため，サーバを複数置き計算機間に階層的な関係を構成することが考えられる．完全L分木維持問題は，計算機の停止や再起動が起こるたびにこのような関係を常にバランスを保つように修正することに対応する問題である．

第5章の結論では，本研究で得られた主な結果をまとめ，今後に残された問題について述べている．

# 関連発表論文

[1] 齊藤，都志，辻野，都倉：“高速 LAN のモデルとその上での更新アルゴリズム”，電子情報通信学会研究報告，COMP88-14, (1988-05).

[2] 齊藤，都志，辻野，都倉:”高速 LAN のモデルとその上での更新アルゴリズムについて”，電子情報通信学会論文誌（Ｄ Ｉ）,J72-D-I, 2, pp.108-116,1989.

[3] 齊藤，辻野，都倉:” 高速 LAN モデルにおける生成木維持問題”，電子情報通信学会研究報告， COMP89-22,(1989-06).

[4] 齊藤，辻野，都倉:”完全生成木維持アルゴリズムとその複雑度”，電子情報通信学会論文誌（採録決定）.

# 故障を考慮した高速LANのモデルと分散アルゴリズムに関する研究

## 目次

# 第 1 章

# 緒論

最近，イーサネット [1]，トークンリングなど伝送速度や遅延が計算機内部でのデータ転送とそれほど（数分の 1〜十数分の 1 程度）変わらない高速な LAN (ローカルエリアネットワーク) が普及してきた．その結果，高速 LAN による分散環境下で複数の計算機が協調して動作するようなソフトウェアに対する要求が高まっている．このようなソフトウェアのためのアルゴリズムは，「ある問題を解くために必要な情報がネットワーク上の各計算機に分散している状況で，ネットワーク上の各計算機が，メッセージのやりとりで情報を交換して，その問題を解く」というアルゴリズムであるととらえることができる．ネットワーク上でのプログラム (アルゴリズム) をこのような枠組みでとらえて行われた研究としては，分散アルゴリズムに関する研究がある．

しかしそこで一般に用いられているネットワークモデルや評価基準は，point-to-point の通信路 (リンク) によって計算機が相互に接続されたネットワークを想定したものである (図 1.1)．これらは広域ネットワークを意識したものである．メッセージの送受信は，通信リンクの端点に対応づけられたポートにメッセージを送り込んだり，取り出したりすることで行なう．これまでに各種の分散アルゴリズムが提案され，アルゴリズムの計算複雑度等が研究されている [2]．

分散アルゴリズムの研究で用いられるネットワークのモデルは同期式と非同期式に大きく分類される．同期式ネットワークは，メッセージの遅延時間や計算機の動作速

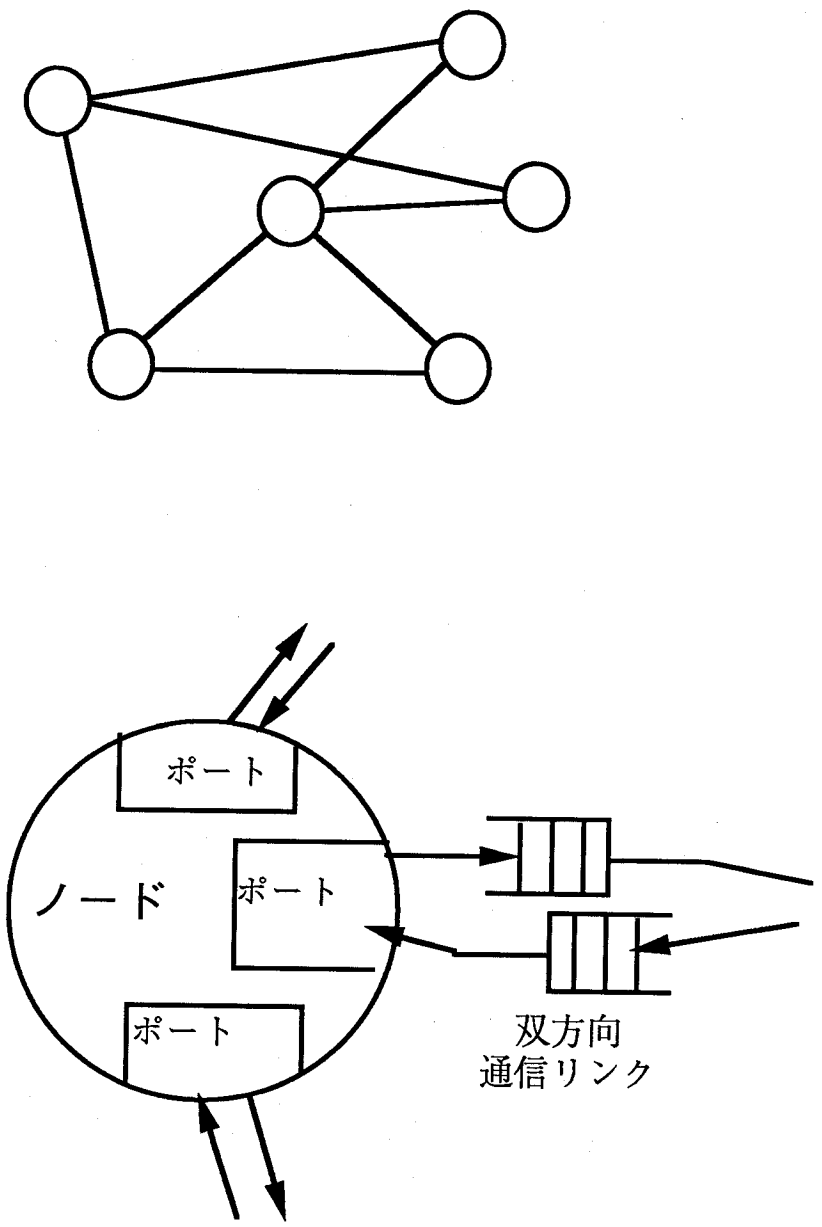



図 1.1: ノードとリンクからなるネットワークモデル

度の比に何らかの上限が存在するネットワークモデルである．それに対して非同期式ネットワークモデルでは，故障がない場合には，メッセージは失われずにいつか通信リンクの他端に届けられることが保証されるが，通信遅延の上限がない．そのため，あるポートにメッセージが着信しない場合，単にメッセージが遅れているのかあるいは送られていない，あるいはリンクが故障しているのかが区別できない．既知の分散アルゴリズムの多くは非同期式ネットワークに対して提案されている．

分散アルゴリズムで扱われる故障は，停止故障とビザンチン故障に分類される．停止故障は通信リンクや計算機が全く働かなくなるものである．通信リンクはメッセージを伝えなくなり，計算機は一切メッセージを発信しない．一方ビザンチン故障は正しく働かなくなるもので，不正なメッセージを送信することもあり得るとするものである．その他，メッセージの消失を含んだ故障についての研究もなされている．本論文では停止故障だけを考える．

故障の取り扱い方は，故障がどのように起こるかによっても分類できる．

- 故障が存在するネットワークが初期状態として与えられ，アルゴリズム始動後の新たな故障発生はないもの，
- 故障が複数箇所で時間をおいて発生し得るもの，
- アルゴリズムの動作中に故障発生があり得るもの，
- 故障が発生するだけでなく，一度停止したものが後で再起動するもの，

などが考えられる．

非同期式ネットワークのもとでは，アルゴリズムが動き始める前にすべての故障は起こって，固定している，すなわちアルゴリズムの動作中に新たに故障は発生しないとすることが多い．非同期式ネットワークでは，メッセージの遅延と故障が区別できないため，故障を取り扱うことが困難である．故障した計算機が存在するネットワークでその形状について予備知識がない場合，コンセンサス問題†を解くアルゴリズム

†各計算機が何種類かの相異なる初期値を与えられ，アルゴリズムが終了する時にすべての正常な計

は存在しないことが知られている．また非同期ネットワークではアルゴリズムの始動後に故障が新たに起き得る状況は扱うことが出来ない．

一方，実際のネットワーク環境を考えると，故障は複数回時間をおいて起こり得るし，また故障が起こってもそのままということは少なく，いつか修復されて再び動作を始めると考えられる．また，ある故障に対応するために動いているアルゴリズムが完了するまえに別の故障が発生することもあり得る．また故障に対応するために，それ起こる前にあらかじめ何らかの予備動作を行なうことは可能であり，また有効だと考えられる．しかしそのようなアルゴリズムを作成するためには，アルゴリズムの始動後に故障が発生する状況を扱うことが必要である．

複数の計算機が協調して動作する場合，ある計算機が停止した時に協調動作が停止せず残りの計算機だけで働くことは重要である．しかし，長時間連続してネットワークを稼働させることを考えると，停止した計算機が再起動した場合に，その計算機を再び協調動作の仲間に入れることが必要不可欠になる．

本論文では高速 LAN での状況を想定し，故障と回復 (再起動) の両方がありうる状況での分散アルゴリズムについて考察する．

LAN 環境での分散アルゴリズムを考える場合，ネットワークのハードウェアを操作する低レベルのアルゴリズムからアプリケーションプログラムレベルのアルゴリズムまで広い範囲が考えられる．本論文では，高速 LAN の上でのアプリケーションプログラムのレベルでのアルゴリズムについて考察する．そこで，アプリケーションレベルのプログラムから見た高速 LAN の性質を反映したネットワークモデルについて考察する．

LAN を利用する時にはネットワークの物理的構造，メッセージどうしの競合回避操作などは，ハードウェアや下位のプロトコル階層によって行なわれて見えなくなっており，低レベルの操作に煩わされる事なくアプリケーションプログラムを構築できる．またその地理的規模の小ささゆえに，以下のように広域ネットワークよりも有利

---

算機が同じ値を出力値として持っているようにする問題．リーダ選択問題より易しい．

な特徴を持っている[1].

- どの計算機とでも通信できる.

  point-to-point リンクでネットワークを構成する場合，直接交信可能な計算機はリンクで結ばれた計算機だけである．LAN ではネットワーク中のどの計算機にも直接メッセージを送ることが出来る．ソフトウェアから見た論理的な LAN をノードとリンクからなるネットワークモデルで表そうとすれば，ネットワークの形状は完全グラフとなる.

- 形状は固定している

  ノードとリンクからなるネットワークでは，通信リンクの追加や削除によりネットワーク全体の形状は大きく変化する．例えばネットワーク形状が完全グラフであるとき，どの計算機とでも直接通信できることに依存したアルゴリズムは作成可能であるが，リンクが削除されると動作しなくなってしまう．一方 LAN では，計算機の追加やケーブルの延長が行なわれてもソフトウェアから見た論理的なネットワークの姿に変わりはない．そのため，どの計算機とでも直接通信が可能であることに依存したアルゴリズムを作成しても差し支えない.

- ある計算機が停止しても他の計算機どうしの通信には影響を与えない.

  当然停止した計算機への通信はできないが，ある計算機の停止によって動き続けている計算機どうしの通信が阻害される事はない.

- 通信コストが安く，均一である.

  通常，LAN では通信に課金する事はない．また通信するための計算機に与える負荷や，遅延時間は通信相手によらず均一である.

- 通信速度が早く，伝送遅延時間が小さい.

  これによって，送受信の同時性が生まれる．またタイムアウトなどの手法が使用可能になる.

- 各計算機は停止したり起動したりする.

  障害による計算機の停止の頻度は無視できるほど低くはない. また停止した計算機はそたままとは限らず, 自動的に再起動する事が多い.

- 障害は主に計算機に起こる.

  ネットワークを構成する計算機と比べて, 通信路 (ケーブル) の故障は非常に少ない.

本論文では以上のような高速 LAN の性質を反映した仮定をおいたネットワークモデルを提案することで, 動的な故障に対処する分散アルゴリズムを作成するための枠組を提供し, その上での問題について考察する.

上で述べたように, LAN においては一つの計算機の停止は, 他の計算機どうしの通信には影響を及ぼさないのが普通である. ところが現実にあるプログラムでは, ある一つの計算機をサーバにしている事が多い. こうすると, プログラムは簡単になるが, サーバの計算機が停止すると他のすべての計算機に影響が及ぶ. そこで, ネットワーク全体としての機能に注目し, 一部計算機の停止がおきても残った計算機である機能を果たし続ける事について考える.

本論文の第 2章～第 3章は, 関連発表論文 [1]～[2] として公表した研究をまとめたものである.

第 2 章では, 高速 LAN の性質を反映したネットワークモデルを提案する.

一時的な機能の停止と再起動が繰り返し起こる状況で何らかの機能を維持する場合, アルゴリズムは終了することが出来ない. 行なうべき動作を完了した場合には, 停止や再起動など動作を始める必要を生じる事象が次に起こるのを待つ待機状態にはいる必要がある. そこで, 定常状態, 受信待ち状態, 待機状態について定義している.

ノード (計算機) とリンクによるネットワークモデルでは, ネットワークの状況を表す尺度として連結度や直径など, ネットワークの形状に対応したグラフに関する尺度が用いられる. 一方本論文で提案するモデルではどの 2 台のノードも直接通信が可能

ではあるが，双方が同時刻に動作している必要がある．そこで，停止と再起動の発生する形態に注目して，そのネットワークの結合性の尺度を定義している．

第 3章では，情報更新問題を解くアルゴリズムを作成し，正当性を証明する．情報更新問題とは，ネットワーク上のどのノードからも更新要求が発生し得る状況のもとで，最後に発生した情報をすべてのノードに配る問題である．ここでは，同時に動作することがないような 2 台のノードが存在するほど悪い状況のネットワークでも，第 3 者のノードで中継することによりこの問題を解くアルゴリズムが構成できることを示している．

本論文の第 4章は，関連発表論文 [3]～[4] として公表した研究をまとめたものであり，完全 L 分木維持問題とそれを解くアルゴリズムおよびその評価を示す．

ネットワーク環境において多数の計算機に整合性のある動作をさせるため，計算機間の関係に何からの構造を作ることは良く行なわれている．そのもっとも単純な例は 1 台のリーダ計算機とそれ以外のスレーブという構造であり，リーダを選出するためのリーダ選択分散アルゴリズムに関しては多くの研究が行なわれている．リーダが 1 台だけ存在する状況では，他のすべての計算機がリーダと通信して情報交換を行なうと考えられる．LAN の規模が大きくなるとリーダの負荷が問題となってくる．そこで，整合性を保つための要となるリーダ (マスターサーバ) のほかにスレーブ計算機からの情報照会等に答えるための中間階層の計算機 (スレーブサーバ) を置く例が良く見られる．このような場合，スレーブサーバに均等にクライアントを割り当てることが望ましい．またマスターサーバやスレーブサーバが停止した時には代役となる計算機を選出して置き換える必要が生じる．このような，マスターサーバ－スレーブサーバ－クライアント 階層において，あるノードが通信する一つ上の階層の計算機はどれか一つに定まっていることがほとんどである．よって各ノードと一つ上の階層の計算機との対応関係は木構造としてとらえることが出来る．4章で扱う完全 L 分木維持問題は 1 台のノードが直接サービスできる相手が最大 L 台である時に，バランスのとれた階層的なサーバクライアント関係の構造を維持する問題である．

一般に，ネットワーク全体としての動作の（機能の）維持を実現する時二つの手法が考えられる．

(1) 停止や起動が起こるたびに最初に戻ってすべての計算をやり直す

(2) 以前の計算結果を利用し，停止や起動の影響を受ける部分だけを再計算する

前者の手法をとる場合，ネットワークの始動時に実行する，構成問題に対するアルゴリズムを必要に応じて再実行すれば良い．後者ではアルゴリズムの性能が改良できる可能性があるが，維持を行なうアルゴリズムを別に作成する必要がある．

4章で示すアルゴリズムはノードの停止や起動の影響を受ける部分の再計算を効率良く行なわせるための付加情報を各ノードに持たせている．この時問題になるのは，ノードの停止や起動に対応して完全L分木を修正するにつれ，付加情報も修正する必要がある点である．付加情報を持たせることによる効率向上分を付加情報の更新のための計算量が越えないように，バランスのとれたアルゴリズムである必要がある．

分散アルゴリズムの評価において，メッセージ数は代表的な尺度であり，通信料金に対応するものである．高速LANにおいては通信コストが小さいため，各ノードの内部での計算も無視できない．分散アルゴリズムのもう一つの代表的な尺度である理想時間計算量はアルゴリズムの修了までの所要時間に関連した量であるが，メッセージの遅延による時間だけに注目している．高速LANにおいては通信遅延が小さいため，ノード内での計算の所要時間は無視すべきでない．

分散アルゴリズムのある一つの計算機に注目した動きと通常の逐次アルゴリズムのもっとも大きな違いは，分散アルゴリズムの実行所要時間には通信待ち時間が含まれることである．逐次アルゴリズムの評価においては，計算量は計算時間を表すものである．4章で使用する局所計算量は各計算機でのアルゴリズムの動きを，通常の逐次アルゴリズムと類似の基準で評価したもので，通信操作の重みを $O(1)$ としている．よってこの局所計算量はそれぞれの計算機での計算負荷に対応する．メッセージの送受信には少なくともメッセージ数に比例した局所計算量を要するため，メッセージ数

が有効な尺度であることに変わりはない．メッセージ１個に付随して起こる処理の局所計算量が $O(1)$ ならば，メッセージ数と局所計算量の総量のオーダーは同じになる．

4章で示すアルゴリズムは１つの修復あたりの局所計算量の総和の最悪値が小さくなるように工夫して構成されている．このアルゴリズムはマスタサーバから末端の計算機までのサーバ階層が h 層である状態で停止あるいは起動が起こった時，完全 L 分木維持問題を最悪時 $O(L+h)$ メッセージで解く．ここで h= $\lceil \log_L((L-1)$(動作中の計算機数)$+1)\rceil -1$ である．このアルゴリズムでは局所計算量の総和は $O(h+h\log L+L)$ となる．局所計算量のもっとも大きいノードでの計算量は $O(L)$ である．

# 第 2 章

# 高速 LAN のモデル

本節では高速 LAN を想定したネットワークモデルについて述べる.

## 2.1 高速 LAN の性質

高速 LAN は一般に以下に示すような性質を持つ. 特に性質 (1 ), (2) のため, 一般に用いられている分散アルゴリズムのネットワークモデルがうまく適合しない.

(1) 通信遅延時間が小さい. また, 途中で長時間の蓄積をともなう中継が行われない. このため,

- ある一定時間以内に通信が成功しない場合にはタイムアウトで通信失敗とみなすようなプロトコルを採用できる. 相手がメッセージを受信したことをもって送信の成功とする (送受信の同時性) 場合もある.
- 通信遅延時間が小さいため, アルゴリズムの時間評価において, ローカルな計算処理時間が無視できない.

  通信遅延と計算機内でメッセージあたりのローカルな処理時間には大きな差はない. 磁気ディスクのアクセスなどを伴った場合には, むしろ通信路での遅延の方が小さくなる.

(2) 通信のコストが小さい.

一般的な分散アルゴリズムのネットワークモデルでは，通信が従量課金でまた計算処理コストよりも高価であることを想定し，通信量に注目してアルゴリズムの評価を行なっている．一方高速 LAN では，ケーブルを敷設してしまうと以後の通信コストは非常に小さい．また，LAN ではメッセージの送信に対して課金することは普通行なわれない．実際，LAN 環境では通信量は通信路の能力を超過しない限り問題とはされず，通信に関する各計算機での負荷がよく問題になっている．

(3) 物理的には通信路は共用であり，故障は非常に少ない.

(4) ある 2 つの計算機は，他の計算機の状態に関係なく，双方が動作中であれば通信できる.

(5) ネットワーク上の計算機は，そのネットワークに属する全計算機の識別子を知っている.

(6) 計算機の停止や回復が日常的に起きる.

利用者の誤操作による一時的なシステムダウンや，夜間の停止などのように停止し，その後回復するのが普通である.

LAN の高速性は性質 (3) と (4) は密接に関わっている．バス型にせよリング型にせよ，物理的に見ればある瞬間に通信媒体を使用可能な計算機は 1 台 (もしくはせいぜい数台) である．しかし，媒体を操作するハードウェアとネットワークプロトコルの下位層の働きにより，上位のソフトウェアからみれば任意の計算機と直接通信できる論理的なネットワークが実現されている．ある 2 ノードが直接通信可能であることをノード間にリンクが存在することで表すなら，LAN で提供される論理的なネットワークの形状は完全グラフとなる．完全グラフ構造はあくまでも論理的なものなので故障に関しては通信路の故障は全体的である．特定の 2 計算機間の通信だけができなくな

ることはない．通信路の障害はほとんど起こらないことと，起こったとしたら通信が全く行なえなくなり有効な障害回避手段はないと考えられるので本論文では通信路の故障は扱わず，ネットワークに接続した各計算機の故障だけを考慮する．

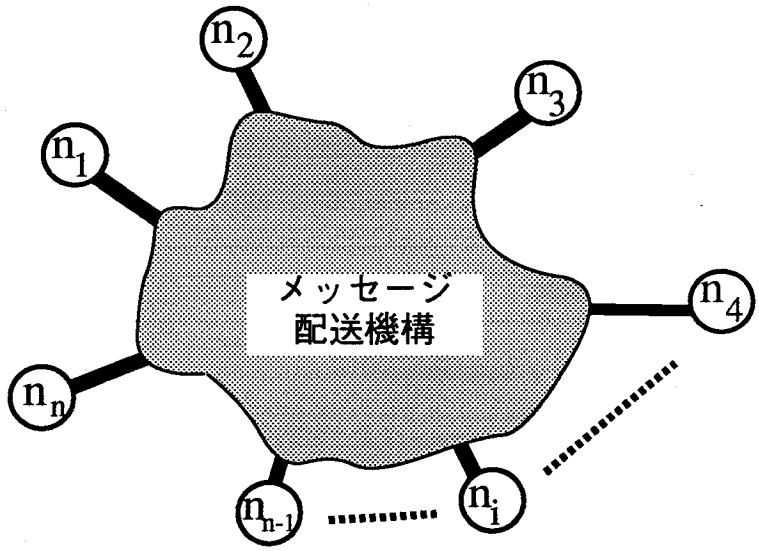
## 2.2 モデルの設定

2.1 節の考察にもとづいて作成した高速 LAN のネットワークモデルを以下に示す．複数のネットワークをゲートウェイを介して接続したインターネットでは性質 (4) は成り立たなくなるが，ここでは単一のネットワークを想定している．

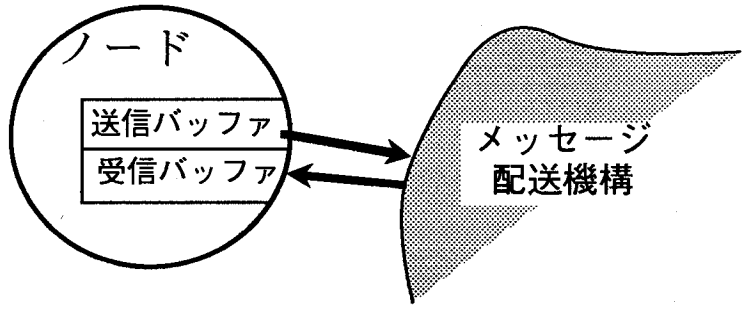
**定義 1 (ネットワークとノード)** ネットワークはノードからなり，ノードはメッセージの交換によってのみ通信することができる．ノードとは，計算機やプロセッサ・エレメントを抽象化した呼び名である．

ネットワーク N はノード $n_1, n_2, \ldots, n_n$ の集合として定義する．すなわち，N = $\{n_1, n_2, \ldots, n_n\}$, □

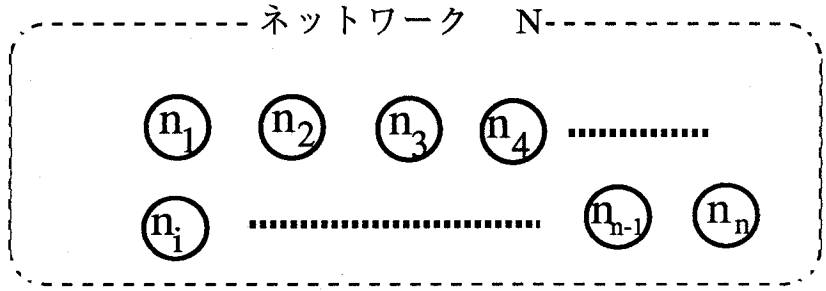
このモデルにはネットワークの構成要素としてリンクに関する定義がない．これは LAN にはもともと point-to-point のリンクはないことと，性質 (3), (4) を反映したものである．高速 LAN に対するイメージは図 2.1(a) のようなものである．この図でのメッセージ配送機構の働きは実際には高速 LAN のハードウェアとそれを駆動するソフトウェアによって提供されるものである．図 2.1(b) のように送受信バッファを仮定することもあるが，これも機能の説明のために用いる概念である．ネットワークをメッセージ配送機構 D を含めて，例えば N= $\{D, n_1, n_2, \ldots, n_n\}$ と定義することも可能ではあるが，メッセージ配送機構は常に 1 つだけ存在し，すべてのノードが接続しているものなので，名前をつけたりネットワークモデルの構成要素として陽に記す必要はない．モデルは単純であることが望ましいので，ネットワークモデルの構成要素はノードだけとした．




(a) ネットワークのイメージ




(b) ノードのイメージ




(c) ノードの集合としてのネットワーク

図 2.1: 本論文でのネットワークモデル

このモデルでは定義8によりすべてのノードが相互に通信可能としている．このような状況をリンクの存在によって2ノード間を相互通信可能とするモデル[(2)]で表すならば，完全グラフ形状のネットワークになる．またリンクを用いるモデルではメッセージを送信する操作はポートにメッセージを入れることであり，送信操作の完了はメッセージの相手先への着信とは独立の事象である．メッセージが届かないうちに次のメッセージを同じポートに投入することも許される．通信ポートが宛先毎に個別に存在すると仮定すると，各ノードへの送信が並列に行なえると仮定するのが自然である．一方本論文のモデルではLANの特徴を反映するために，(定義4, 仮定5) 送信操作は相手にメッセージが着信するまで完了せず，また複数の送信操作を並列には行なえない．このため個別のポートの存在を仮定することは不自然である．

また例えば3つの計算機a,b,cを考える．aがメッセージをまずbに続いてcに送ったとする．そしてaからのメッセージを受けとった結果，cがbにメッセージを送ったとする．リンクをFIFOとしてとらえるネットワークモデルでは，同期ネットワークでも非同期ネットワークでも，bにaからのメッセージが先につくことは保証されなず，cからのメッセージが先につくことも考慮してアルゴリズムを構成しなければならない．一方LANでは，cからのメッセージが先にbにつくことは考えにくい．ノード間にリンクを仮定したネットワークモデルでは，この性質を持たせることが難しい．

これらの理由で，ネットワークの構成要素はノードだけとしてポートやリンクなどの具体的な網の構成要素については規定せず，単にすべてのノードと通信できる等の機能レベルでの定義を行なった．

**定義 2** メッセージ長は最大メッセージ長 (定数) 以下の可変長とする．最大メッセージ長は $O(\log n)$ ビット程度とする． □

現実のネットワークではパケットの大きさには制限があり，1000～数万ビット程度である．これを越える大きさのデータの送受は複数のパケットに分割して送受される．また，アルゴリズム (ユーザプログラム) が送るメッセージに，宛先アドレスや送り元アドレスなどの情報が付加されたものがネットワークに送り出されるパケットとなる．

このモデルでは，最大パケットサイズを越えない程度のメッセージを使用するアルゴリズムを考える．メッセージの最大長を規定することで，アルゴリズムの評価の際にメッセージ数をパケット数と対応づけて考えることが可能となる．

**仮定 1** 各ノードはユニークな識別子をもつ．またネットワークに存在する全ノードの識別子と自分自身の識別子を予め与えられているものとする． □

**定義 3** ある時刻 t におけるノード n の状態を S(t,n) $\in \{run, stop\}$ で表わす．

- 状態が stop であるとき停止中であるという．
- 状態が run であるとき動作中であるという．
- 状態が run から stop になることを停止するという．
- 状態が stop から run になることを回復するという． □

**仮定 2** 各ノードはアルゴリズムの状態 (変数値，制御状態など) の他に不揮発性記憶を持つ．ノードが停止すると，アルゴリズムの状態は失われるものとする．そのため，あらかじめ与えられていた情報と不揮発性記憶に格納しておいた情報以外の，回復して以後獲得したすべての情報は失われる． □

この仮定は計算機が停止した時点でのプログラムの変数値は失われ，計算機が再起動する時にはプログラムも初期状態で起動されることに対応している．

不揮発性記憶はディスクファイルを想定したものである．計算機が停止した場合，実行中のプログラムの変数値などの揮発性記憶 (主記憶上の変数などを想定) は失われる磁気ディスク装置に保管した情報は保存される．ただし，不揮発性記憶は格納並びに参照等に要するコストは揮発性記憶に比べて大きいものと考える．

**仮定 3** 各ノードは，非同期に停止，回復する．それがいつ起きるかは予測できない．
□

定義 3と仮定 2,3は，性質 (6) を反映している．

**定義 4** ノードはプリミティブ send と receive を用いて通信する．メッセージの宛先は，識別子で指定する．またメッセージを受けとると，その送りもとノードの識別子も分かる．また任意の 2 ノードは相互に通信可能とする．

- 送信 send(mes,dst) : (suc,fail)(mes, dst はともに値引数)

  ノード dst にメッセージ mes を送る．

  あて先ノードが receive プリミティブでメッセージを取り込んだなら，suc(成功) を返す．一定時間 $T_W$ 以内に成功しなければ，失敗として fail を返す．

  $T_W$ はネットワーク上のノード数によって定まる．この $T_W$ を最大送信待ち時間と呼ぶ．

- 受信 receive(mes,sender) (mes, sender はともに出力引数)

  メッセージを受けとるプリミティブである．

  どこかのノードからメッセージが送られて来るまで待ち，mes に送られてきたメッセージを,sender に送信元ノードの識別子をいれて返す．

□

(注) これらのプリミティブは性質 (1 ) を反映して決めたもので，それらの内部では，タイムアウト，再送などの処理が行なわれていることを想定している．$T_W$は下位プロトコルによって規定されているタイムアウトの時間である．実際の LAN ではノード数に上限があるので $T_W$は定数であるが，ここではネットワークのノード数に依存し，O(ノード数) だと考える．

LAN も含め，メッセージ喪失の可能性がある媒体の上での信頼性のある (メッセージの消失がない) メッセージ交換プロトコルは，一般的には Acknowledge 返送，タイムアウト，メッセージ再送などを組み合わせて実現される．メッセージの宛先の計

算機の停止を考慮するならば，メッセージ再送を無制限には行なわず，ある時点で送信操作を中止するであろう．送信操作を開始してから再送をとりやめるまでの時間を$T_O$とする． LAN では媒体上でメッセージが保管されることがない．このことから，もしメッセージが相手に届くなら，送信を開始してからの時間は$T_O$未満であり，またメッセージの着信を送信側が知るのは，メッセージが着信した後になる．定義 4 での$T_W$は$T_O$に対応するものである．現実には，送信側での再送が規定の回数に達し送信をあきらめた場合，メッセージが届いていない可能性だけでなく，メッセージが届いているが返送された Acknowledge が規定の時間までに届かなかった可能性もある．最後の送信再試行からの待ち時間の設定などが適切であれば後者の状況は起こらないはずである．このモデルでは後者の状況は起こらないものとしている．

**仮定 4** すべてのノードで同じアルゴリズムが実行される． □

本論文ではアルゴリズムは PASCAL 風のの言語で記述する．

**仮定 5** メッセージの送受信は同期して行なわれる．そのため，ノード a が b にメッセージを送った場合， a の送信開始，b の受信，a の送信完了がこの順に起こる． □

**定義 5 受信待ち状態**: あるノードが受信待ち状態にあるとは，そのノードに受信済みのメッセージがなく，かつアルゴリズムの制御が receive プリミティブ内にある状態をいう． □

**定義 6 待機状態**: あるノードのアルゴリズムが待機状態にある時，そのノードは待機状態であるという．

アルゴリズムの待機状態は，受信待ち状態のうちの特殊なもので，アルゴリズム毎に定義される．動作中であって待機状態以外のノードの状態をまとめて稼働状態と呼ぶ．
□

**定義 7 定常状態**: ネットワークが定常状態とはネットワークの動作中のすべてのノードが待機状態にある事をいう． □

これは，通常の逐次アルゴリズムにおける終了に対応するものである．

## 2.3 ネットワークの結合性

ネットワークがノードと通信路からなるとするモデルでは，ネットワークの状況はノードを頂点，リンクを辺に対応させたグラフの形状や連結度で評価される．しかしこのモデルでは任意の 2 ノードが通信できるが，それは双方が同時に動作しているときに限る．よって本論文で提案する高速 LAN のモデル上での分散アルゴリズムにとってのネットワークの結合性は，ノードが同時に動作している時間帯がどのように分布するかによって評価すべきである．

ネットワークとその上のノードについての結合性を記述する述語として次の四つを定義する．

**定義 8** ノード $n_1$,$n_2$について connect($\tau$, k, $n_1$, $n_2$) を以下のように定義する．

connect($\tau$, k, $n_1$, $n_2$)

$$iff$$

$\forall t(\tau < t < \tau + k \cdot T_W)$ :

$$(S(t, n_1) = run \wedge S(t, n_2) = run)$$ □

時刻 $\tau$ から少なくとも k 回連続して通信できる時間，両方のノードが動作中である ( $S(t, n_i) = run$ である) こと．connect($\tau$, k, $n_1$, $n_2$) であることを時刻$\tau$で $n_1$, $n_2$ の間に通信できる機会があるともいう．

**定義 9** ノード $n_1$,$n_2$ について node_connect(t,k,$n_1$,$n_2$) を以下のように定義する．

node_connect(t,k,$n_1$,$n_2$)

$$iff$$

$\forall T, \exists \tau(T < \tau < T + t)$ :

$$connect(\tau, k, n_1, n_2)$$ □

任意時刻 T から時間 t の間に，connect である時刻が少なくとも一回はある．すなわち，いつでも最悪 t 以内に通信可能になることが保証されている状況を表す．

**定義 10** ネットワーク N について strong_connect(t,k,N) を以下のように定義する.

strong_connect(t,k,N)

$$iff$$

$\forall n_1, n_2 \in N$ :

$$node\_connect(t, k, n_1, n_2)$$ □

ネットワーク N 上の任意のノード $n_1$, $n_2$ のあいだに node_connect の関係が成り立つことである.

**定義 11** ネットワーク N について weak_connect(t,k,N) を以下のように定義する.

$$weak\_connect(t, k, N) \ iff$$

$$\forall n_1, n_2 \in N$$

$$node\_connect(t, k, n_1, n_2) \vee$$

$$\left( \begin{array}{l} \exists T, \ \exists m_1, m_2, .., m_p, \tau_1, \tau_2, .., \tau_{p+1} \\ \left( \begin{array}{l} T \ < \tau_1 < \tau_1 + k \cdot T_W < \tau_2 < \ldots \\ \quad < \tau_i < \tau_i + k \cdot T_W < \tau_{i+1} < \ldots \\ \quad < \tau_{p+1} < \tau_{p+1} + k \cdot T_W < T + t \end{array} \right) : \\ \quad connect(\tau_1, k, n_1, m_1) \quad \wedge \\ \quad connect(\tau_2, k, m_1, m_2) \quad \wedge \\ \quad \ldots \quad \wedge \\ \quad connect(\tau_{p+1}, k, m_p, n_2) \end{array} \right)$$

□

任意の $n_1$から $n_2$へ時間 t 以内に, 直接またはいくつかのノードを経由してメッセージを送る経路及び機会がある.

(注) この $m_1, m_2, \ldots, m_p$ を, 通信経路と呼ぶ.

**仮定 6** ネットワーク N は, ある t と k について weak_connect(t,k,N) を満たす. □

ノードとリンクからなるネットワークモデルでは，少なくともネットワークが連結であることが仮定されている．非連結のネットワークでは他のノードのどのように中継を受けてもメッセージの伝達が出来ないノードが存在し得るからである．本論文のモデルでは，weak_connect がこれに対応する．

ネットワークのもっと有利な状況とは，すべてのノードが動作し続けて，決して停止が起こらない状況である．この状況では相手の停止のために送信が失敗することはない．ノードの停止が起こり得る状況の中でも，strong_connect は比較的有利な状況を表す．ノードの停止が起こりえるとき，アルゴリズムは送信に失敗した場合に対応する必要がある．その方法としては，まず停止した相手が再起動するのを待って改めて送るという手法が考えられる．strong_connect は，このような簡単な手法が使用可能であるようなネットワークの状態をあらわしたものである．

weak_connect が成り立たないようなネットワークの状況とは，直接間接を問わず，メッセージを届けることができない 2 ノードが存在するということである．たとえば，計算機が昼間のみ稼働するもの (A 群) と夜間のみ稼働するもの (B 群) の 2 群に分かれているような極端な状況を考えてみる．このような状況では，ある計算機が他群の計算機にメッセージを伝えることは不可能であり，strong_connect(t,k,N), weak_connect(t,k,N) ともにどのような t,k に対しても成り立たない．そのため全計算機が協調して働くようなアルゴリズムを作成することは不可能である．

ここで，昼から夜あるいは夜から昼にかけて動作し続ける第 3 種 (C 群) の計算機が存在するとする．C 群の計算機を仲立ちとすれば，A 群と B 群の計算機は間接的にではあるが情報の交換が可能である．

strong_connect(t,k,N) はこの状況でも成り立たないが weak_conenct(t,k,N) は成り立つ．ある計算機の持つ情報を他の計算機に伝える必要があるときに，単純に直接送信を試みるだけのアルゴリズムは不適当であるが，他のノードを経由して情報を伝えるアルゴリズムならネットワーク中の全計算機が協調できる可能性がある．

すなわち，weak_connect が成り立つことはこのモデル上で，全計算機が参加する

ような分散アルゴリズムを作成し得るためのネットワークの状況に対する最低限の条件である.

## 2.4 結合性の例

上で定義した結合性の尺度 strong_connect と weak_connect を例を挙げて説明する.

全ノードが 24 時間運転しているような LAN を考える. 故障による長期間の停止はなく, 誤操作やソフトウェアのエラーによるなどによる一時的なシステムダウンはあるものとする. また各ノードはシステムダウンの後は必ず再起動し, 15 分以内に回復するものとする. また $K \cdot T_W$は 15 分と比べて十分に小さいものとする.

- strong_connect

  めったに停止が起きない (連続稼働時間が 15 分と比べて十分に長い) とする. 二つのノードの停止期間が重なることがあり得ないほど停止が希であれば, strong_connect(15 分, k,N) である. あるノード $n_1$が $n_2$に送信を試みた時点で $n_2$が動作中ならば, その時点で通信が成立する. $n_2$が停止中だとしても, その時点から最悪でも 15 分以内には回復する.

  二つのノードの停止期間が重なることがあり得る場合を考えると, このネットワーク N は, strong_connect(30 分, k, N) であるといえる. 図 2.2 のように, あるノード $n_1$が $n_2$に送信を試みる直前に $n_2$が停止し, $n_2$が回復する直前に $n_1$が停止した場合が最悪となる. $n_1$が停止した後, 最悪でも 15 分後には $n_1$が回復するので, $n_1$が $n_2$に送信を試みた時点からの通算では 30 分で通信が成功する.

- weak_connect

  ノードの停止が頻繁に起きるが, 全ノードが同時に停止する事はないという状況を考える (図 2.3). たとえ, $n_2$にメッセージを送ろうとした $n_1$が, $n_2$が回復したときには停止してしまっていた場合, 直接送ることはできない. 二つのノードが交

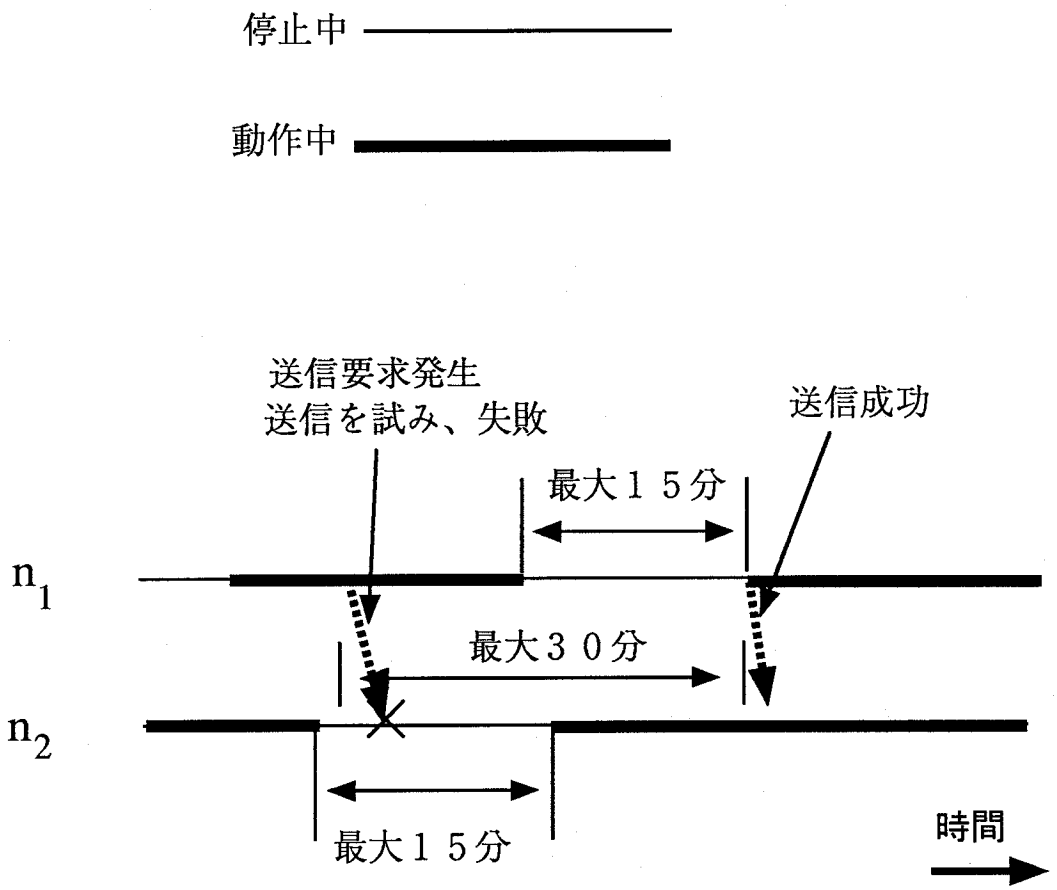



図 2.2: strong_connect の例

互に停止するという極端な場合を考えると，strong_connectは成立しない．しかしどれかのノード(この例ではノード$n_4$)に中継してもらえば，$n_2$の回復時に$n_1$からのメッセージを届けることができる．このネットワークはweak_connect(15分，k，N)である．

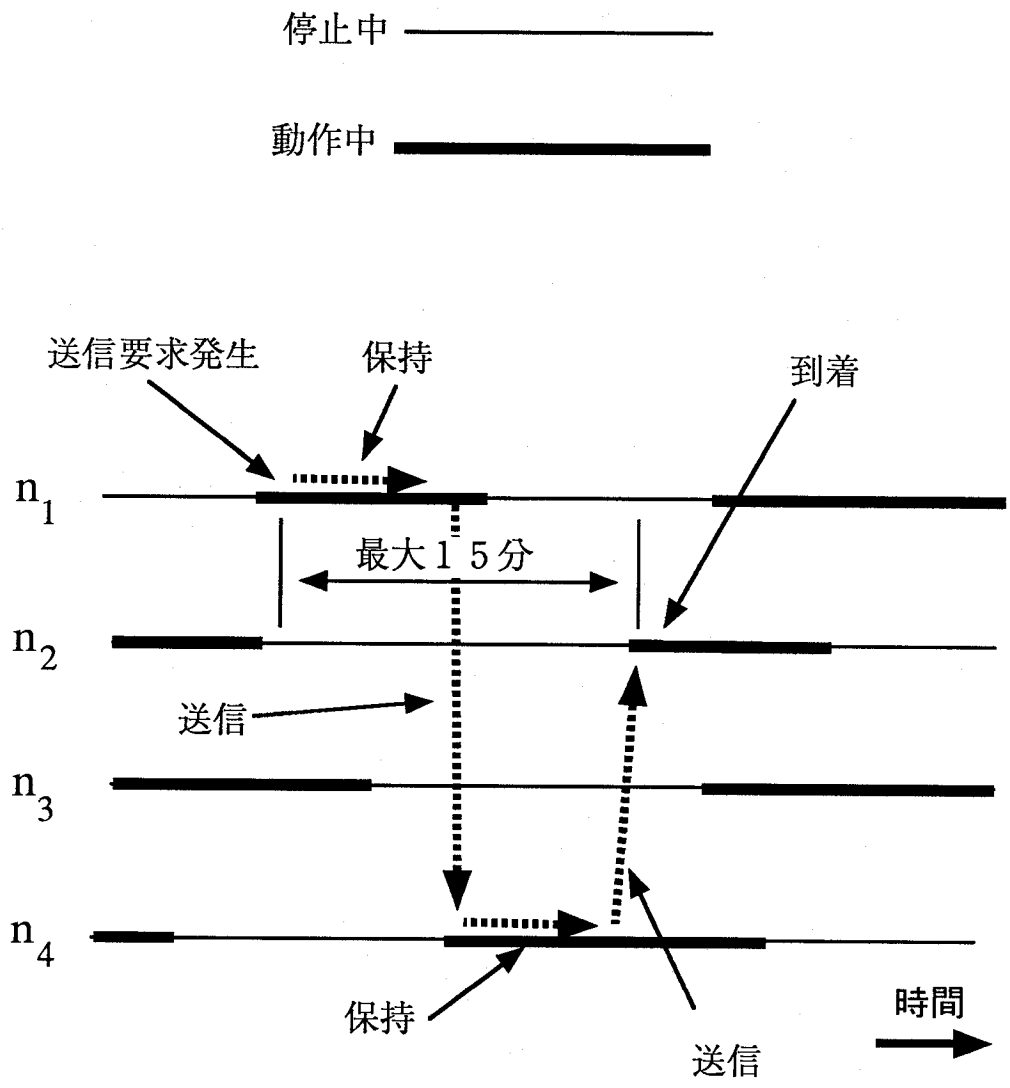



図 2.3: weak_connect の例

# 第 3 章

# 情報更新問題とそれを解くアルゴリズム

## 3.1 情報更新問題

2章で示したLANのモデル上で，情報更新問題を解くLANアルゴリズムを考える．

ある情報のコピーをすべてのノードに配る問題はブロードキャスト問題と呼ばれ分散アルゴリズムの研究においてもっとも基本的な問題の一つである．ここで，ブロードキャスト操作が複数回繰り返し行なわれることを考えてみる．また配布する情報の発生源も特定のノードとは限らずどのノードも配布元になり得るとする．このとき，配布すべき情報の発生ではなく，新たに発生した情報で以前の情報を更新するものと考える．この場合は最新の情報さえ持っていれば良いので，ある情報の配布が完了しないうちに別の情報の配布が始まる場合，先に始まった配布作業はとりやめて良い．

本論文では故障中 (停止中) の計算機 (ノード) もいつかは回復することに注目し，モデルに組み込んでいる．故障したノードの扱いに関しては，単に無視し，動作中のノードの間だけで情報の更新が行なわれれば良いとすることもできる．しかし本章で扱う情報更新問題ではさらに，停止していたノードが再起動した時に，その時点での最新情報を伝えることも問題に含める．

情報更新問題に関連する現実の問題の一例としてはネットワーク環境でのパスワード管理問題がある．これは全計算機に設定されているパスワードを同じ内容であるようにすること†と，任意の計算機からパスワードを変更することを実現する問題である．パスワードの場合は，利用者の数だけ項目があるのに対し情報更新問題では簡単のために1項目だけとするが，問題の本質は変わらない．

### 3.1.1 問題の定義

**目的**：ネットワーク上の全ノードが同じ情報をもっていなければならない項目(配布対象情報)がある．ネットワーク上の全ノードで，非同期に配布対象情報の更新要求が起き得る．配布対象情報が更新されたとき，ネットワーク上の全ノードにその新しい配布対象情報を届けて更新させ，全ノードが同じ最新の配布対象情報をもつようにする．

各ノードでの配布対象情報の更新履歴が異なってもかまわない．より新しい情報が発生した時点で，それ以前に発生した情報の配布は放棄してよいものとする． □

情報更新問題とは上記の目的を実現するアルゴリズムを2章で示したモデルの上で，以下に示す条件(1)〜(5)のもとで作成する問題である．

(1) あるノードでの情報更新要求の発生間隔は $T_W$ 以上であるとする．

無制限に頻繁に更新要求が起きることを許すと，情報更新問題を解くことができないのは明らかである．よって更新要求の発生頻度に関して何らかの仮定をおかざるを得ない．

(2) ネットワークの結合性の仮定は最低限にとどめる．すなわち weak_connect(t, k, N) だけは満たす．

(3) 情報発生直後に計算機が停止した場合，その情報が失われることは許す．

†実在のネットワークサービスである NIS(Network Information Sercies) [3]ではこの問題とは対照的に「同一のパスワードで複数の計算機を利用する」という目的をパスワードを配るのではなく，必要が生じるたびにサーバに問い合わせることで実現している．

このモデルでは，ノードの停止はいつでも起き得る．よって発生した情報を不揮発性記憶に格納する前に停止した場合には，その情報は失われるし，これを回避する方法はない．よってこの条件をおく．

(4) 分散環境でのイベント発生時刻の順序づけに関しては，既に研究がなされている[4] [5]．そこで異なるノードで発生した更新要求の時間順序の判定は情報更新問題とは切り離して考える．つまり，新旧判定の方法はあるものとしてアルゴリズムを作る．

(5) 停止したノードは少なくとも $T_W$ の間回復しない．

ノードの停止回復もある一定の頻度以下であると仮定する．回復したノードは必ずある一定時間は停止しないという仮定は不自然である．よって停止している時間の最小値を仮定することで停止回復の頻度を制約する．

注) (2) での t は与えられた条件であるが，k はこの問題を解くアルゴリズムによって定まるものである．全ノードが同じ情報を持つのに必要な時間が短いほど，また k の下限が小さいほどよい解であるといえる．

## 3.2 更新問題を解くアルゴリズム

### 3.2.1 考察とアルゴリズムの概略

この問題の重要な点を考察する．

- weak_connect であるから情報の発生したノードから他のすべてのノードへ時間 t 以内に情報を伝える経路と機会の存在は保証されている．しかしその経路がどういうものか予め分かる事は保証していない．そこで，この経路を捜して，確実に利用することが問題となる．

- この問題では，停止しないか，あるいは他のノードと比べて停止しにくい特定のノードの存在は仮定できない．役割を持つノードは作らず，全ノードが対等

に働くアルゴリズムが望ましいと考えられる.

- ノードの停止はいつでも起き得るので，いつ停止してもよいようにアルゴリズムを構成する必要がある.

以上の点に留意して，情報更新問題を解くアルゴリズムを作成した．各ノードでの処理をイベント駆動形式で書くと，以下のようになる.

1. ローカルな更新要求が発生したら，自分の持つ配布対象情報を更新する.
2. 新しい配布対象情報を受信した場合にも，自分の持つ配布対象情報を更新する.
3. 配布対象情報が更新されたら，他の全ノードに配布対象情報を送信する.
4. 受信した配布対象情報が自分が持つのと同じであったら，無視する.
5. 回復時には，全ノードに (不揮発性記憶に保管してあった) 配布対象情報を送信する.
6. 古い配布対象情報を含むメッセージを受信したら，その発信元ノードに自分の持つ配布対象情報を送信する (返送処理).

新しい配布対象情報の発生源のノードから他のノードへの経路を考える．weak_connect より，この経路に沿って情報を伝達すれば，時間 t 以内に情報を伝えることができる．しかし経路上のあるノード $n_j$は，次のノード $n_k$ が分からない．どれかはわからない経路上の次のノードに確実に情報を伝えるには，全ノードに送信すればよい．上の処理 1)〜4) でこれが行われる.

次に，weak_connect で保証された機会とは，$k \cdot T_W$以上の期間 $t_j$と $t_k$ が動作する期間である．この期間の始まりについて考えると，$n_j$で情報が発生したとき，$n_j$が経路での一つ前のノードから情報を受信したときあるいは，$n_j$, $n_k$ どちらかが回復したときである．$n_j$の回復で通信できる機会が始まるのなら，処理 5 でこの機会を利用できる．機会が $n_k$ の回復で始まるのなら，処理 5 で $n_k$ が送ったメッセージが $n_j$に届き，処理 6 に従って $n_j$が $n_k$ に情報を送信するのでこの機会もうまく利用できる.

### 3.2.2 アルゴリズム全体の構成

アルゴリズムをリスト 3.1 に示す．以下のプリミティブや仮定を用いている．

**仮定 7** 各ノードは自分の識別子を変数 myname, ネットワーク上の全ノードの識別子の集合を変数 net として参照できるものとする． □

**仮定 8** ローカルに発生する情報を取り出すプリミティブを get(info) とする．get(info) 情報更新要求がローカルに発生するのを待ち，発生した情報を info に格納する． □

**仮定 9** 各ノードはマルチプロセス方式をとり，複数のプロセスが並行して処理を行なうことができる - 複数のアルゴリズムが同一ノード内で並列に実行される - ものとする． □

アルゴリズムをマルチプロセス方式で記述するのは，複数の非同期な事象を取り扱うアルゴリズムを平易に記述するための便法である．「メッセージ受信かローカルな情報発生かどちらかが起こるまで待つ」などといったプリミティブを定義すればマルチプロセスにせずにこのアルゴリズムを記述することが出来るが，記述が複雑になり，分かりにくくなる．そこで，マルチプロセス形式でアルゴリズムを記述した．

**仮定 10** 待ち行列は共有変数であり，プロセス間の通信に利用できるものとする．待ち行列操作のプリミティブは以下のものが利用できるものとする．

- procedure enqueue(data:QUE_ENTRY)

  待ち行列の末尾にデータを一つ入れる．

- procedure dequeue(var data:QUE_ENTRY)

  待ち行列の先頭からデータを一つ取り出す．データがない場合には，データが入って来るまで待つ．

- function empty_queue():boolean

  待ち行列の中にデータの有無を調べる．

このアルゴリズムで使用する待ち行列は一つだけなので，待ち行列を識別するための引数は省略する． □

以下にアルゴリズムの主要な特徴を列挙する。

- このアルゴリズムで使用するメッセージの種類は単一であり，送信元ノードの持つ配布対象情報を含む．
- このネットワークモデルでは，ノードが動作していても，メッセージが送信されたときに receive プリミティブを実行していなければそれを受け取ることができない．また，配布その他の作業中にメッセージの受信や情報の発生などが非同期に起きる．そこで図 3.1 に示すように，受信したメッセージと発生した情報は，一つの待ち行列に入れ，入力イベントを単一にする．
- 受信処理プログラムは， receive の実行と受信した配布対象情報の待ち行列への格納を繰り返し行う．
- 発生処理プログラムは，更新要求の発生を待ち，発生した配布対象情報を含むメッセージを作成し，それを待ち行列に入れる．
- その結果，配布更新処理プロセスにはローカルに発生した情報は自分自身からのメッセージに見える．
- 回復時には，待ち行列の初期化などの初期化処理が行われ，受信処理，発生処理，配布更新処理プロセスがそれぞれ起動される．
- 各ノードは，配布対象情報を不揮発性記憶に格納するが，同時に変数 myinfo(揮発性記憶) にも格納するものとする．参照は常に myinfo に対して行われる．

**仮定 11** 不揮発性記憶の読み書きにはプリミティブ read_nv,store_nv が利用できるものとする．

```
procedure read_nv(var inf:MESSAGE);
```

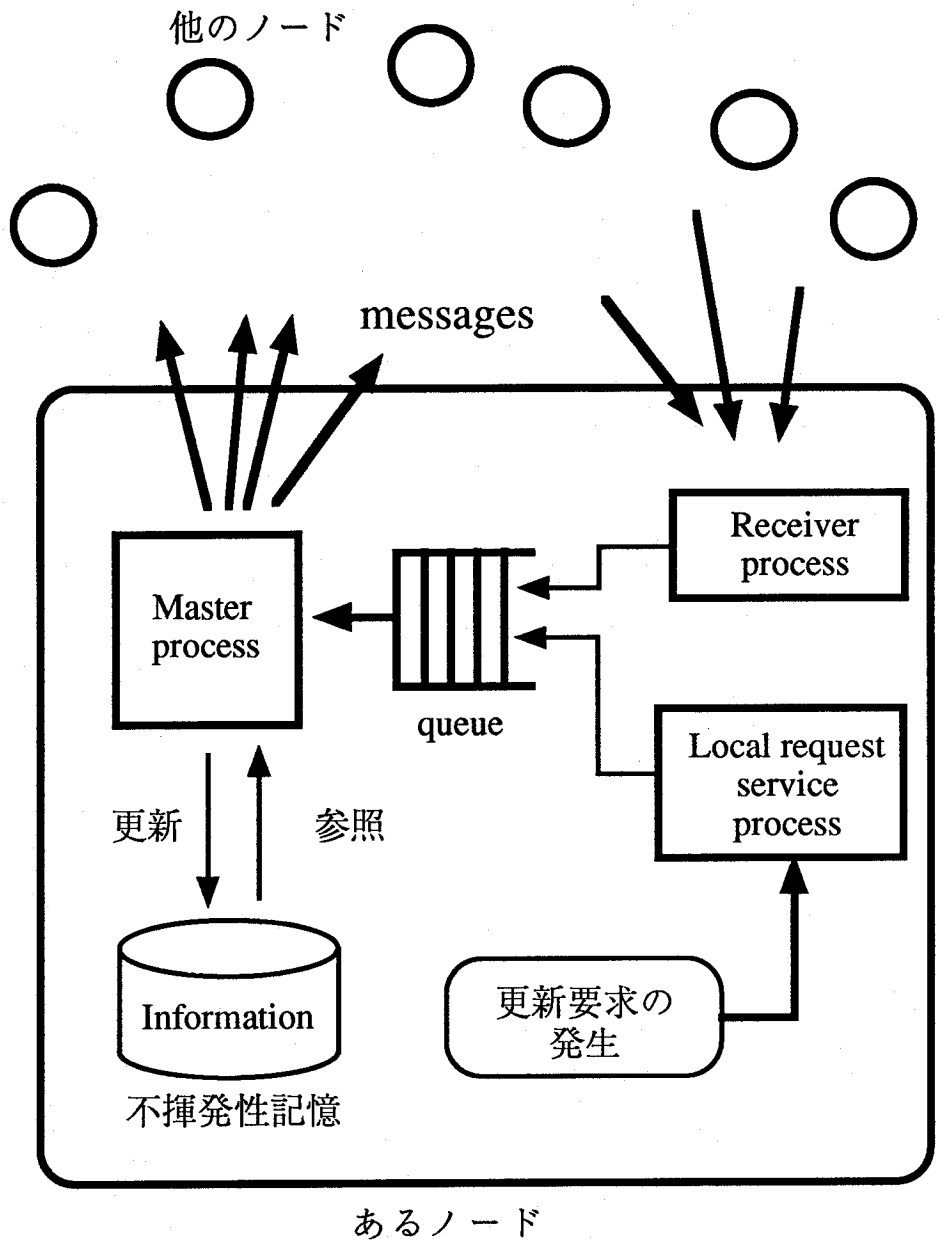

他のノード
messages
Receiver process
Master process
queue
Local request service process
更新
参照
Information
更新要求の発生
不揮発性記憶
あるノード

図 3.1: プロセス構成

procedure store_nv(inf:MESSAGE);

これらの実行に要する時間は高々$T_f$であるものとする．プリミティブの実行途中にノードが停止した場合でも，不揮発性記憶は完全に書き変わっているか，あるいは古い情報のままままったく変化していないかのいずれかであるとする． □

アルゴリズムはすでに述べた通りであるが，重要な点について補足説明する．

メインルーチン変数 range は送信の対象のノードの識別子の集合である．自分を除く全ノードをこれに設定することにより配布を行う．range に含まれるノードへの送信を始めたら，1 回送る毎に待ち行列を調べ，空でなければ先に待ち行列内のメッセージの処理を行う．その結果更新が起これば実行中の配布を取りやめ，新たに配布を始める．古い情報を受信した場合 変数 range に入れておき，返送処理は現在の送信対象への送信が終了するまで行わない．

## 3.3 アルゴリズムの評価と検討

### 3.3.1 評価のための諸定義

アルゴリズムの評価を行なうために，いくつかの定義を行なう．まず記法について定義する．

**定義 12** n：ネットワーク上の全ノード数 (一定)
時間：

- $T_f$：不揮発性記憶への格納に要する時間
- $T_C$：不揮発性記憶の更新に要する時間を除いた 1 メッセージあたりの最大計算処理時間． □

1 メッセージあたりの処理時間は高々$T_C + T_f$となる．

**仮定 12** $T_W > nT_C, T_W > T_f$ と仮定する． □

定義 4の注より $T_W > nT_C$は妥当な仮定である．また $T_W > T_f$ を仮定しないと，ローカルに起きる更新要求にすら対応できなくなる ( 発生間隔 $> T_W$).

このアルゴリズムの正当性を証明するにはまずネットワーク上の全ノードが，同じ最新の配布対象情報を持つようになる (アルゴリズムの完全性) ことを示す必要がある．また更新問題では常に新たな情報の発生を待つ必要があり，アルゴリズムが停止しない．そこで，ネットワークの定常状態を定義し，発生や回復がなければある一定時間内にネットワークが定常状態となることで停止性に代える．

**定義 13** このアルゴリズムでは，以下の条件がすべて成り立つ状態を待機状態と定義する．

- 待ち行列は空である．
- 受信処理プロセスは受信待ち状態である．
- 発生処理プロセスは get プリミティブによる更新要求の発生待ち状態にある．
- 配布更新処理プロセス dequeue プリミティブによる待ち状態にある．

□

定義 6, 7, 13より，動作中の全ノードで待ち行列が空で，その全プロセスが send プリミティブ以外での待ち状態にあるときがこのアルゴリズムが各ノードで実行されているネットワークの定常状態となる．

### 3.3.2 アルゴリズムの完全性

まず証明に必要な補題と系を示す．

**[補題 1]** 休止状態のノードは，回復したときに保持していた配布対象情報と回復以後受信した情報と回復以後そのノードで発生した情報のうち，最新のものを保持している． □

(証明) 受信した情報と発生した情報は待ち行列に格納され，失われることはない．また，配布処理プログラムは待ち行列から取り出した情報が新しいときはかならず更新を行う．休止状態のノードは，待ち行列が空であり，発生及び受信した情報はすべて配布処理プログラムによって処理されている．よって補題 1 は成り立つ． □

**[補題 2]** 新情報発生がない長さ $m \cdot T_W$ の期間に，あるノード $n_i$が動作し続けたとすると，その間に起きる $n_i$にあてた送信は高々$(2m + 3n + 2)(n - 1)/2$ 回，回復するノードがない場合には高々$3n(n - 1)/2$ 回である． □

(証明)

(i) 情報の発生により行われる $n_i$への送信は起きない．

(ii) 回復したノードは他のすべてのノードにメッセージを送信する．あるノードは時間間隔 Tw に高々1 回しか回復しない．よって $T_W$ごとに $n_i$以外のすべてのノードが停止，回復したとしても $n_i$への送信は m$T_W$あたりたかだか m(n-1) 回である．

(iii) $n_i$の回復時に行う配布処理がこの期間に含まれていたとする．$n_i$より新しい情報を持つノードは，その情報を $n_i$に返送する．他のすべてのノードが $n_i$より新しい情報を持っていたとしても，この送信は n-1 回しか起きない．よって $mT_W$ のあいだに，n の回復時処理の結果ノード $n_i$に返送されるメッセージはたかだか (n-1) である．

(iv) 各ノードが持つ配布対象情報の種類は高々 n 種類である．あるノードがより新しい配付対象情報を受信する回数は，すべてのノードが異なった情報を持っていたときが最大で，

| | |
|---|---|
| 1 番新しい情報を持つノード | 0 回 |
| 2 番目に新しい情報を持つノード | 1 回 |
| ⋮ | |
| 1 番古い情報を持つノード | $n - 1$ 回 |

となる．よって新しい情報の受信による配布で，ノード $n_i$あてに送信されるのは高々n(n-1)/2 回である．

(v) $n_i$が更新の結果行う配布の結果，より新しい情報 $inf_j$を持つ $n_j$ は $n_i$に情報を返送し，その結果 $n_i$の情報は更新される (この返送のための send プリミティブ実行中にもう一度 $n_i$メッセージが届いたとすると，それに対しても返送処理を行う)．それ以降 $n_i$が送信するメッセージは，$inf_j$より古いことはないので，$n_j$から $inf_j$を $n_i$に返送するのは高々2 回．(iv) と同様にすべてのノードが異なった情報を持っていて，$n_i$がいちばん古い情報を持っていたとしても，$n_i$への情報の返送は，高々$(1+2+...(n-2)+(n-1)) \times 2 = n(n-1)$回．よって (i)〜(v) より新しい情報の発生がない期間に，動作し続けるあるノード $n_i$にあてて起きる送信の回数は，$m \cdot T_W$の間に高々(2m+3n+2)(n-1)/2 回，発生も回復もなければ (3n+2)(n-1)/2 回である． □

補題 2 から次の系が得られる．

**[系 2]** 発生がない期間には，待ち行列のなかのデータの個数は，(3n+4)(n-1)/2 をこえない． □

(証明) このアルゴリズムでは待ち行列が空になるまで送信は行わず受信したメッセージの処理に専念する．よって，送信開始時には待ち行列は空である．1 回の送信は高々$1 \cdot T_W$で終了するので，この間に受信するメッセージは高々$(2 \cdot 1 + 3n + 2)(n-1)/2$個である． □

次に，配布処理に関する補題を示す．

**[補題 3]** アルゴリズムにおけるある配布対象情報に関する一回の (全ノードに対する) 配布 (メインルーチンの for each 文) はたかだか $K_B \cdot T_W$以内に終了する．ここで，$K_B = (n-1) + \frac{5n(n-1)}{2(T_W/T_C - n + 1)}$である． □

(証明) 新しい情報を受信した場合には配布は中断するので，考えない．アルゴリズムにおける一回の配布では送信は高々 (n-1) 回行われる．配布に要する時間を $x \cdot T_W$とする．受信するメッセージ 1 つあたりの処理時間は高々$T_C$．補題 2 より，$x \cdot T_W$の

間に受信するメッセージはたかだか (2x+3n+2)(n-1)/2. 送信とメッセージの受信処理の時間合計は高々$(2x+3n+2)(n-1)T_C/2+(n-1)T_W$であるから

$$\begin{aligned} xT_W &\geq (2x+3n+2)(n-1)T_C/2+(n-1)T_W \\ x(T_W-(n-1)T_C) &\geq (3n+2)(n-1)T_C/2+(n-1)T_W \end{aligned}$$

仮定 12より，$T_W-(n-1)T_C>0$ であるから両辺をこれで割って，

$$\begin{aligned} x &\geq \frac{(3n+2)(n-1)T_C+2(n-1)T_W}{2(T_W-(n-1)T_C)} \\ &= \frac{2(n-1)T_W+(5n-2n+2)(n-1)T_C}{2(T_W-(n-1)T_C)} \\ &= \frac{2(n-1)T_W-2(n-1)(n-1)T_C+5n(n-1)T_C}{2(T_W-(n-1)T_C)} \\ &= \frac{2(n-1)(T_W-(n-1)T_C)+5n(n-1)T_C}{2(T_W-(n-1)T_C)} \\ &= (n-1)+\frac{5n(n-1)}{2(T_W/T_C-n+1)} \end{aligned}$$

よってたかだか $(n-1)+\frac{5n(n-1)}{2(T_W/T_C-n+1)}$以内に一回の配布は終了する. □

**[補題 4]**weak_connect で保証された経路で $2K_BT_W+(3n+4)(n-1)T_C/2+T_f$ 以上の動作期間の重なりがあれば情報が経路に沿って伝わる. □

(証明) $n_i$が最新情報を持っているとする. $n_j$との通信機会の始まりが,

(a) $n_i$が inf を受信あるいは発生した時である場合.

系 2 より受信後高々$(3n+4)(n-1)T_C/2$ でその情報を待ち行列から取り出し，不揮発性記憶に書き込み $(T_f)$, 配布を始める. 補題 2 より，配布は $K_BT_W$で完了するから，高々$(3n+4)(n-1)T_C/2+T_f+K_B\cdot T_W$ 以内に $n_j$へ inf を送信する.

(b) $n_i$が回復したときである場合.

補題 2 より，$n_i$は $K_B\cdot T_W$以内に配布を完了し，$n_j$に inf が届く.

(c) $n_j$が回復したときである場合.

補題 2 より，$n_j$は $K_B \cdot T_W$以内に配布を完了する．$n_i$が配布中であったなら，$K_B \cdot T_W$以内に配布を終了し，返送処理を開始する．返送先ノードの数は高々$n-1$よって配布と同じく高々$K_B \cdot T_W$以内に返送処理を完了する．$n_j$は，$(3n+4)(n-1)/2 \cdot T_C$以内に $n_i$からのメッセージを待ち行列から取り出し不揮発性記憶に書き込む．合計で高々$2K_B \cdot T_W + (3n+4)(n-1)/2 \cdot T_C + T_f$.

(a)－(c)より (c) の場合が最も大きな動作時間の重なりを必要とし，それは $2K_B T_W + (3n+4)(n-1)/2 \cdot T_C + T_f$. □

**[定理 1]** ネットワーク N が weak-connect(t,k,N) で，$k > 2K_B + \frac{(3n+4)(n-1)/2 \cdot T_C + T_f}{T_W}$ならば，あるノード $n_1$で情報が発生してから後，ネットワーク内で新たな情報の発生がないまま時間 t が経過すれば，inf をネットワーク上の全ノードが持っている． □

(証明) $n_1$で発生した情報は最新であるので，他のノードに届けば，既に届いている場合を除いてそのノードはそれを持つようになる．補題 4 より $kT_W$の幅の機会があればそれは確実に利用できるので，高々時間 t 後までにはどのノードにも最新情報が届きかつ更新が完了している． □

### 3.3.3 アルゴリズムの停止性

前節では，アルゴリズムによって全ノードが同じ情報を持つようになることを示した．この節では，アルゴリズム実行によって，ネットワークが定常状態にはいることを示す．

**[定理 2]** ある情報が発生した後，新たな情報の発生や回復がなく，すべてのノードが時刻 T にその最後に発生した情報を保持していたならば，高々 $2K_B \cdot T_W + (3n+4)(n-1)/2 \cdot /T_C$ の後には定常状態になる． □

(証明) 全ノードがある時刻 T に同じ配布対象情報を持ったとする．T 以降送信されるメッセージはすべて最新の情報を含むので，単に無視される．T で配布中であるノー

ドは高々$K_B \cdot T_W$以内に配布を終了する．その後Tまでに既に受信していたメッセージに対する返送処理を行うがこれも高々$K_B \cdot T_W$以内に完了する．よってTから高々$2K_B \cdot T_W$以内に最後の送信が終わる．この時点で待ち行列が保持しているメッセージは系2よりたかだか(3n+4)(n-1)/2．よって時刻Tから$2K_B \cdot T_W + (3n+4)(n-1)/2 \cdot T_C$後にはどのノードでも，待ち行列は空になっている．すなわち定常状態となる． □

定理1，2よりこのアルゴリズムは正しい．

リスト 3.1

情報更新問題を解くアルゴリズム

リスト3.1 情報更新問題を解くアルゴリズム 1/2

```
(a)共通に使用するデータ型
type
        NODE_ID=(ノードの識別子)
        INFO=〈通信する配布対象情報の型〉
        QUE_ENTRY=record        〈待ち行列のデータの型〉
                sender:NODE_ID;〈配布対象情報の送信元〉
                mes:MESSAGE;            〈配布対象情報〉
        end
COMPTYP=(NEWER,OLDER,SAME);〈情報比較     compareの型〉

(b)受信処理プロセス
program    receiveproc;         〈受信したデータを格納する〉
        var    entry_data:QUE_ENTRY;    〈queue に格納するデータ〉
begin
        while (true) do begin 〈常に受信待ち〉
                receive(entry_data.mes,entry_data.sender);
                enqueue(entry_data);    〈データを格納する〉
        end
end.

(c)発生処理プロセス
program    local_request;
var    entry_data:QUE_ENTRY;       〈queue に格納するデータ〉
begin
        while (true) do begin 〈常に発生待ち〉
                get(entry_data.mes.data);
                        〈発生したデータを格納する〉
                enqueue(entry_data);
        end
end.
```

リスト3.1 情報更新問題を解くアルゴリズム 2/2

```
(d)配布更新処理プロセス
program    management
label    10;
var      dst : MODE_ID;
     range,r : set of NODE_ID;
       myinf : MESSAGE;                    {自分自身の情報}
function  msgprocess:boolean;
        var qtop:ENTRY_DATA;
begin
        repeat
        dequque(qtop);
        case    compare(qtop.mes,myinfo)    of
        NEWER:  begin   {発生時, 受信時}
                store_info(qtop.mes);
                myinfo=qtop.mes;
                range:=net-[myname,qtop.sender];
                msgprocess    :=    true
                end;
        OLDER : range := range+[qtop.sender];
        SAME  : {無視}
        end
        until    empty_queue
end;

begin
        read_nv(myinfo);
        range:= net - [myname]; {回復時の配布}
        while  (true)    do    begin
10:
                while(range <>[]) do begin
                        r:=range;    range:=[];
                        foreach dst in r do
                        if not empty_queue then begin
                                if msgprocess = true then
                                        goto 10;
                                {更新時配布中断}
                        end;
                        send(myinf,dst);
                end;
                dummy:=msgprocess;
        end
end.
```

# 第 4 章

# 完全 L 分生成木の維持問題

## 4.1 根つき完全 L 分生成木の維持問題

本章では停止したノードが存在する時に残った動作中のノードがある状況を維持し続けることを考える.

LAN など全ノードが相互に直接通信可能なネットワークで複数のノードが関係する状況を考えてみると, 全ノードと直接, 均等に通信し続けているわけではない. サーバクライアント関係やマスタースレーブ関係など 2 計算機間の何らかの関係を結んで, 特定少数の計算機とだけ頻繁に直接通信を行なう例は多い.

例えば, 計算機群の時刻を一致させる timed [8]では, 1 つの全体を司るマスターサーバが停止するたびに新たなマスターサーバを選出している. また以前にマスターサーバであった計算機が再始動した場合, スレーブとして参加する. 通常, スレーブ計算機どうしが通信することはなく, 各スレーブはマスターとなっている計算機度だけ通信する. スレーブどうしが通信するのは, マスターが停止した時である. timed の場合には維持すべき状況が 1 台だけのマスターサーバと残りはすべてスレーブという状況なので単にリーダ選択を繰り返すだけで足りている.

ここでもっと複雑な状況を考えてみる. 例えば NIS(Network Information Service†)

†旧名 Yellow Page

(3) では，ネットワーク上の計算機がマスターサーバ，スレーブサーバ，クライアントという 3 層構造になっている．スレーブサーバはマスターサーバから情報の提供を受ける一方，自分の担当のクライアントの情報を提供する．

このような状況を想定して例えばサーバクライアント関係のような 2 ノード間の関係を線で表すことで，ネットワーク全体の状況を図示すると，図 4.1 のように描くことができる．

これを一般化して，多階層のサーバクライアント関係を考える．このような状況で計算機の停止と再起動を考えると

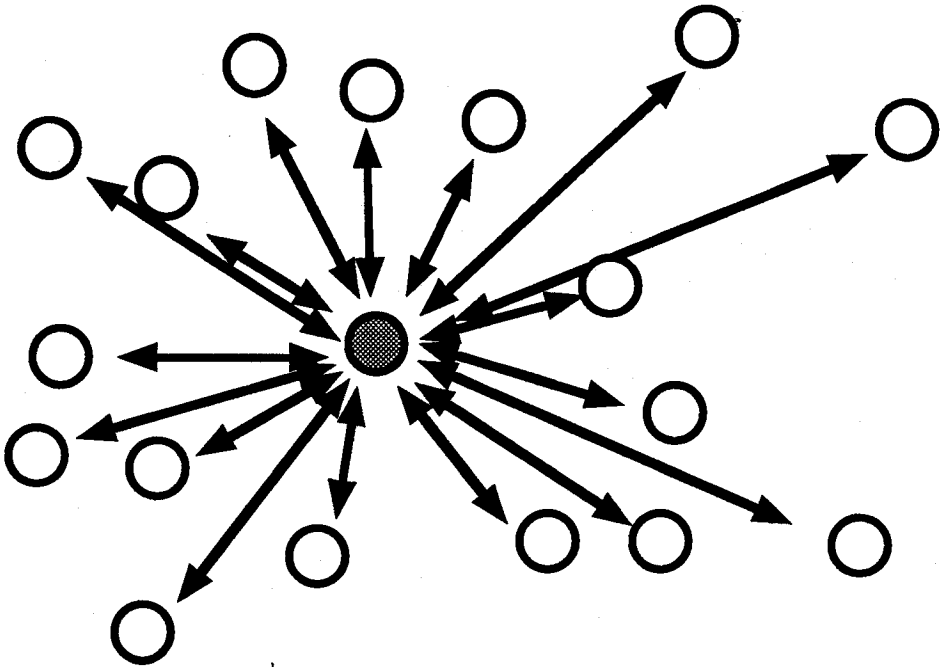
- サーバの停止が起こるとそのサーバが担当していたクライアントを他のサーバに割り振る，
- 計算機が起動するとどこかのサーバのクライアントとする，
- サーバの負荷に偏りが生じないように，混んでいるサーバのクライアントを空いているサーバの担当に変更する，

などの働きが必要となろう．完全 L 分木維持問題とはこのような状況を想定した問題である．

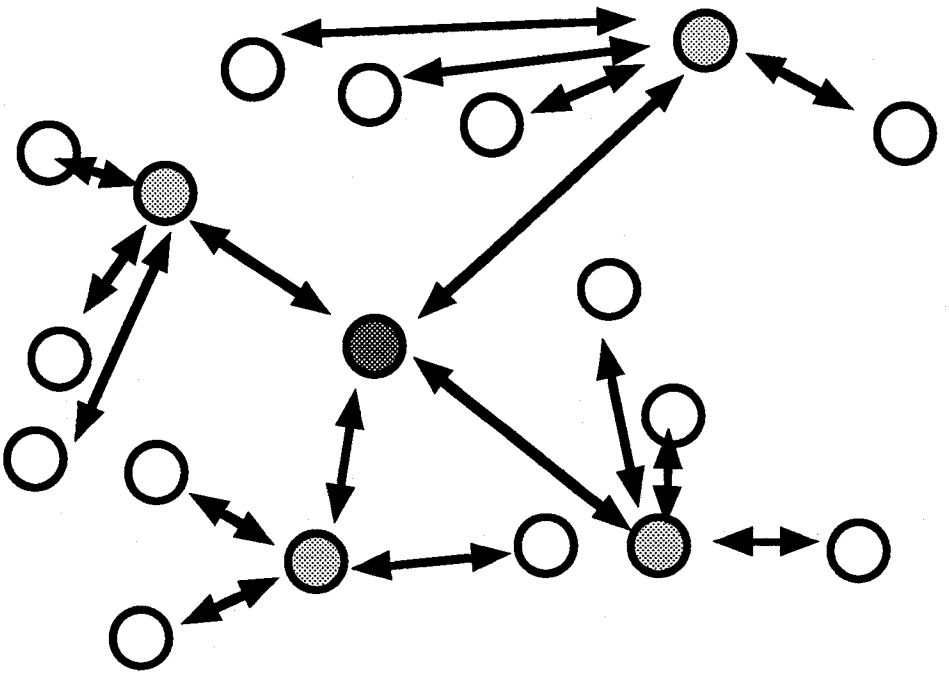
timed, NIS 双方ともサーバ（になり得る）計算機とスレーブ計算機はあらかじめ人間によって決められているが，ここではすべてのノードは均質であり，サーバになり得る能力を持つものと仮定して問題を解く．

計算機と通信リンクで定義されたネットワークモデルにおける生成木構成問題とは，通信リンクを枝，計算機を頂点と考えた時にグラフの形状が生成木になっているように通信リンクの部分集合を求める問題である．生成木の形状に制約をおく例としては，最小重み生成木に関する研究が多く見られる (10)．この場合，辺の重みとは通信リンクの通信コストと対応するものと説明される．

本研究で想定しているようなネットワークはメッセージの伝送コストが小さく，またどの 2 者間でも同じであるような均質なものなので，「最小重み」は意味を持たない．



(a)サーバ(リーダ)への負荷集中



(b)補助サーバによる負荷分散

図 4.1: ノード間の関係を辺で表したグラフ

ここで，ネットワーク上での動作中の全ノードが参加するサーバクライアント関係を考える．サーバが一つだけで，他のすべてのノードはサーバからサービスを受けるものとしてその状況を図示すると図 4.1(a) のようになる．$n$ 個のノード中 $k$個が停止しているとすると，サーバは $n-k-1$ 個のクライアントにサービスすることになる，この図をグラフとしてとらえると，その形状はサーバのノードの対応する頂点を根とする高さ 1 の木となる．

本研究では，各ノードが持つことができる子の数には上限 L があるものとする．L+1 以上のノードを一つの木に納めるためには，ネットワーク上でのサーバクライアント関係にクライアントでありかつサーバでもある中間的な計算機を導入し階層的なものにする必要がある．動作中の全計算機（ノード）がこれに参加すると考えると，サーバクライアント関係をマスターサーバを根とした L 分生成木と捉えることができる．図 4.1(b) は L=4 の場合の例である．図 4.1(a) と同様にグラフとしてみると，4 分木なっている．生成木とは全ノードを含む木だが，ここでは停止中のノードは除外し，存在する全ノードではなく動作中の全ノードを含むものを考える．

このような状況では，各ノードからマスターサーバ (根) までの距離は短い方が良いと考えられる．そこで単なる L 分木でなく完全 L 分木に関する問題を考える．

実際のネットワークで完全 L 分生成木を構成する必要があるのは，停電の後など起動直後のノードしか存在しない時だと考えられる．ネットワークが稼働中に一部のノードが停止したり，停止していたノードが動き出したりした時には，構成アルゴリズムを動かすよりは既存の木の情報を利用して一部を修正するだけで済ませる方が効率的である．生成木が存在する状態でノードの起動や停止が起こった時，生成木を再構成する問題を生成木維持問題と呼ぶ．本研究では完全 L 分生成木維持問題を扱う．ここでいう完全 L 分木とは木の高さを h とした時，レベル h-1 未満のノードはすべて L 個の子を持つような木のことである (図 4.2).

ノードを頂点に，親子関係を有向辺に対応づけたグラフについて考える．本研究では，対応するグラフを木にすることを考える．根，葉，内部ノードなどの用語，木あ

● レベル0以上h未満のノード: L個の子を持つ

● レベルh-1のノード: 子の数は任意

○ レベルhのノード： 子の数は0

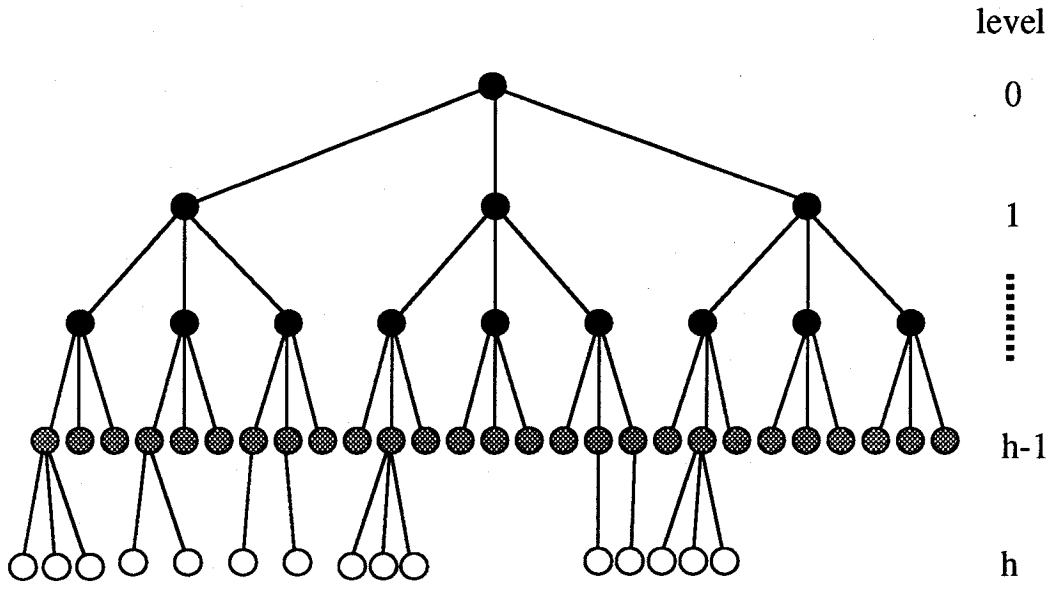



図 4.2: 完全 L 分木

るいは部分木の高さやノードのレベルなどは，グラフ内のノードに対応する頂点に関する，

- 葉ノードの高さは 0,
- 親ノードの高さは子ノードの高さの最大値+1,
- 根ノードのレベルは 0,
- 子ノードのレベルは親ノードのレベル+1,

をそのまま適用する．

根つき完全 L 分生成木の維持すべき性質 **P** は以下のとおりである．h は木の高さとする

**性質 P:**

1. 動作中の全ノードを張る L 分木である (各ノードは生成木の上での自分の 親ノードと子ノードを知っている)．停止中のノードは含まない．
2. レベルが h-1 未満のノードは，子ノードを L 個持っている．
3. 葉ノードはレベル h か h-1 にのみ存在する． □

## 4.2 ネットワークモデル

以下のようなネットワークモデルで問題を解く．

第 2章のモデルでは，複数のノードが同時に相互にメッセージを送信しあった場合などには容易にデッドロックが発生する．このため何らかの手段で送信と並行しての受信を可能にする必要がある．3章では送信や内部処理と並行して受信処理を行なうため，アルゴリズムをマルチプロセスで記述しているが，マルチプロセスモデルを導入するとがプロセス間通信の導入が必要で，モデル，アルゴリズムともに記述が複雑になる．そこで以下の仮定をおく．

**仮定 13** アルゴリズムからは陽には見えない受信バッファが1メッセージ分存在する.

動作中のノードは，アルゴリズムが何を実行しているかにかかわらず，メッセージを1つだけ受信することができる．メッセージを受信したら，アルゴリズムがメッセージを取り出すまでは次のメッセージを受信できない． □

メッセージ送受信のプリミティブは以下のものを使用する.

**send(v,m):** メッセージ m をノード v の受信バッファに入れる.
v が停止中ならば送信は失敗し，エラーを返す．動作中ならば相手がメッセージを受け入れるまで送信は完了しない．送信が完了すると，成功を意味する戻り値を返す.

**receive(v,m):** 受信したメッセージの内容を m に移し，送り元の識別子を v に入れる．受信済みのメッセージがなければメッセージが着信するまで待つ. □

複数のノードが同じノードにメッセージ送信を試みた時，そのノードが送信中だとやはりデッドロックは起こり得る．受信バッファ数を増やすか，送信プリミティブでなく (4) のように送信開始プリミティブを用意する必要がある．ただし本節でのアルゴリズムは仮定 13のもとでデッドロックは起こらないように作られている.

維持問題を解くのアルゴリズムは，ノードの停止や起動が起こるたびに完全L分木になるように修復を行なうものなので終了しない．すなわち，維持問題を解くアルゴリズムは木を修復しても停止せず，次に起こる停止あるいは起動に備えて待機状態にはいる．アルゴリズムとネットワークの状態について以下のように定義する.

**定義 14** ノードの始動
メッセージの受信によらずに稼働状態に移るノードを始動ノードと呼ぶ．始動ノードは一般には複数存在する．この問題の場合，木の修復を始めるノードが始動ノードであり，ノードの停止時にはそのノードの停止を検出したノード，起動時にはそのノード自身が始動ノードとなる．始動ノードではないノードはメッセージの受信で待機状態から稼働状態に移る． □

## 4.3 アルゴリズムの検討

### 4.3.1 完全L分生成木維持問題

3節で述べたモデル上での完全L分生成木維持問題を以下の仮定のもとに解く.

**仮定 14** ノードが起動した場合の始動ノードは起動したノード自身のみである. □

他のノードが, 起動したノードからの直接/間接の通信以外の手段で起動を知ることはない.

**仮定 15** 動作していたノードが停止した場合, 始動ノードは生成木の上で隣接したノードである. 少なくとも1つのノードは始動するものとする. □

このネットワークモデルでは, メッセージの送信プリミティブの戻り値が失敗をあらわすものであることによって, 宛先ノードが停止していることがわかる. 生成木を利用してなにかを行なうときには生成木の枝に沿ってメッセージを送ると考えられるので, この仮定を置く.

**仮定 16** あるノードが起動／停止した後, 生成木の修復を終えてアルゴリズムが完了するまで新たな停止や起動は起こらない. □

すなわち定常状態においてノードが一つだけ起動あるいは停止した場合の木の修復アルゴリズムを考える.

問題を以下のように定義する.

> **P** が成り立っている状態でノードの停止または起動が起こった時, 有限時間以内に **P** が成立しているようにせよ. またネットワークが定常状態になり, 各ノードは次の停止や起動に備えなければならない.

ノードの停止や起動は十分な間隔をおけば (仮定 16), 何回起きても良いようにアルゴリズムを構成しなければならない. すなわち, 維持問題を解くアルゴリズムは木を修復しても停止せず, 次に起こる停止あるいは起動に備えて待機状態にはいる.

### 4.3.2 計算量の評価基準

完全L分木維持問題ではノードの停止や再起動が1つづつ間をおいて起こると仮定する一方，出来るだけ少ない計算量で維持を行なうことを考える．まず，アルゴリズムの計算量の評価尺度について考察する．

一般にはデータの送受信の負荷はデータ量に比例すると考えられる．通信コストを見積もる手法としては，

- メッセージ送受信回数
- メッセージサイズ（ビット）の総量

などが考えられる．前者では，どんなに大量のデータを送信しても1つのメッセージにまとめて送る限り1と数えられてしまう．後者では，小さなメッセージを多数送った場合と大きなメッセージを少数送った場合が区別されない．

現実のネットワークではパケットの大きさには制限があり，1000～数万ビット程度である．これを越える大きさのデータの送受は複数のパケットに分割して送受される．また，アルゴリズム(ユーザプログラム)が送るメッセージに，宛先アドレスや送り元アドレスなどの情報が付加されたものがネットワークに送り出されるパケットとなる．

2.1節で述べたようにLANにおいては通信に対して課金することはふつう行われない．よってメッセージ複雑度やビット複雑度を通信コストの指標とすることは意義がない．しかしながら，通信を行うには送受信双方の計算機にメッセージ処理の計算負荷がかかる．よって，通常の逐次アルゴリズムの評価と同様の計算負荷を想定した計算量に通信計算量を統合して評価することが適当である．そこで以下の点について考察する．

- 個々の計算機での計算量をどのように見積もるか
- 複数の計算機での個々の計算量からどの様に分散アルゴリズムの計算量を求めるか

まず前者について，個々のの計算機でのアルゴリズムの計算量を局所計算量と呼び通常の逐次アルゴリズムと同様に評価する．この時，通信操作のプリミティブをどのように評価するかが問題となる．

定義2より，1メッセージは1パケットに対応すると考える．ここで実際の計算機の通信処理での1パケットあたりの処理の負荷に注目すると，パケット1つあたりの負荷とパケットのサイズに比例する負荷に分離して考えることができる．この二つを比較するとパケット一つあたりの処理負荷が大きい．そこで，複数パケットへの分割が起こらない程度の大きさのメッセージをやりとりするアルゴリズムに関しては，メッセージ送受信操作の実行回数を持ってメッセージ送受信の計算量の指標として扱って差し支えないと考えられる．

ここではメッセージの大きさを $O(\log n)$ 程度としており，ノードの識別子や整数値†などを定数個含んだメッセージが使用可能である．

そこで送受信操作は $O(1)$ の計算量であるとする．実際には受信操作の中には待ち時間も含まれているため，プリミティブの中で経過する時間は $O(1)$ ではない．この評価基準は各ノードでの計算負荷を評価したものと考えられる．

後者については，単純に加算して，局所計算量の総量を求め，これを評価することにする．分散アルゴリズムでは負荷が分散されているか一部の計算機に集中しているかも大きな関心事なので，最も負荷の大きなノードでの局所計算量についても評価する．

局所計算量は，送受信プリミティブにかかる計算量とそれ以外の計算量に分けて考えることができる．ネットワークを通過するメッセージ1個に対し，送信プリミティブは必ず1つ，受信プリミティブは高々1つのノードで実行される．よって，

メッセージ総数 = 送受信プリミティブの実行回数の総和 $\leq$ 局所計算量の総和

という関係が成立する．分散アルゴリズムでは，メッセージ送受信以外の局所計算の多くは受信したメッセージの処理か，あるいは送信するメッセージの作成に要するものだと考えられる．

---

†大きさが $O$(ノード数) を越えないもの

### 4.3.3 修復の戦略

木の維持を行なう手段としては，木を新規に構成するアルゴリズムをそのまま利用することが考えられる．動作中のノード数が n-k の時に，新たに L 分木を作るにはすべての動作中のノードが参加する必要がある．よって，メッセージ数の自明な下界は $O(n-k)$ であり，局所計算量の総和の自明な下界も $O(n-k)$ である．既存の L 分木構造の利用して木の維持を行なうアルゴリズムは，少なくとも $O(n-k)$ よりも少ない計算量で木の修復を行なうことが目標となる．

維持アルゴリズムが動作している状況でノードの停止や起動が何回も起こることを考える．個々の修復の計算量は，アルゴリズムやそのときの木の形状だけでなくその時点までのあるごリズムの動作履歴も関係すると考えられる．ここでは 1 つの修復に要する計算量の最悪値に注目し，これをできるだけ小さくすることを考える．

木の修復は枝のつけ替え，すなわち親子関係の変更を何カ所かで行うことで達成される．ここでアルゴリズムの構成要素を以下のように分類して考える．

1. どのように枝のつけ替えを行うかの戦略
2. 枝のつけ替えの対象となるノードを決める（探す）ための方法
3. 枝のつけ替えそのもののアルゴリズム

まず単純な手法として，以下のようなアルゴリズムについて検討する．

- あるノード v に木全体に関する情報を持たせる．
- v 以外のノードは木に関する情報として
  - v の識別子
  - 親ノードの識別子
  - 子ノードの識別子

だけをもつ．

- 停止や起動が起こった場合には，v に連絡し，v がどの様に木を更新するかをローカルに計算する．そして関連するそれぞれのノードにどの様に修復するかを v が指示する（v は木に関するすべての情報をもっているので，指示するメッセージを送付可能).

このアルゴリズムは通常は小さい計算量で維持を行うが，v が停止した場合が最悪の場合となる．このとき，動作中ノードの中から v の代わりとなるノード v' を撰び，そのことを動作中のすべてのノードに知らせなければならない．少なくとも n-k 個のメッセージが必要であり，局所計算量の総和も $O(n-k)$ 以上となる．

このようにあるノード v が $O(n-k)$ 個のノードに関する情報を持つとすると，v の停止時に $O(n-k)$ メッセージを要する．

同様に，動作中のノードのうちの $O(n-k)$ 個が同じ情報を持っている事が必要であるアルゴリズムは，その情報の更新時に $O(n-k)$ 以上のメッセージと総局所計算量を要する．

これから，多くの情報を少数のノードに集めたり，同じ情報を多くのノードに置いたりするアルゴリズムは 1 回あたりの修復の計算量の最悪値という観点からは良くない事がわかる．

### 4.3.4 アルゴリズムの検討

ノードが起動した時には，既存の木のどこに接続するのが良いかを考える．木の内部に接続してかつ **P** を満たすためには木の中の接続関係を大きく変えなければならない．そこで，葉として接続することを考える．一方ノードが停止した時には，まず木の修復が必要かどうかを判定し，必要な場合のみ修復を行なう．そのため，まずノードの追加や削除に対する **P** を満たす木の性質について考察する．

ここで以下のような記法を導入する．

**R** ：木の根ノード

**T(v)** ：ノード v を根とする部分木

**N(T)** : 木 T のノード全体の集合

**childs($v$)** : ノード $v$の子ノードの集合

**leaf($v$)** : 木 T($v$) の葉ノードの集合

leaf($v$) = $\{u \mid u \in N(T(v)) \wedge \ |childs(u)| = 0\}$.

**parent(r,v)** : 木 T(r) の中でノード v の親ノード.

parent(R,v) は parent(v) と略記する.

**height(v)** : 木 T(v) の高さ

v ∈ leaf(v) ならば, height(v)=0.

v ∉ leaf(v) ならば, height(v) = $max\{height(c) \mid c \in childs(v)\} + 1$.

**h** : 木全体の高さ (height(R))

**level(v,u)** : 木 T(v) の中のノード u のレベル

u = v ならば level(r,v)=0.

u≠v ならば level(v,u)=level(v,parent(v,u))+1.

**levelof(v,l)** : 木 T(v) の中のレベルが l であるノードの集合

$levelof(v,l) = \{u|u \in N(T(v)) \wedge level(v,u) = l\}$.

**spare(v)** : $l = max\{level(u) \mid u \in leaf(v)\}$ として, spare($v$) = $leaf(v) \cap levelof(v,l)$ と定義する. spare(v) は, leaf(v) の中でレベルのもっとも大きなもの集合である. これは $level(v, height(v))$ に等しい.

**fosterer(v)** : T($v$) の中で子の数が L 未満であるノードの集合を f($v$), そのレベルの最小値を $l$ とする. f($v$)=$\{u \mid u \in N(T(v)) \wedge \ |childs(u)| < L\}$, $l = min\{level(v,u) \mid u \in f(v)\}$ である. このとき,

fosterer($v$)= $f(v) \cap levelof(v,l)$ と定義する. fosterer(v) は T(v) の中の子の数が L 未満のノードのうち, レベルがもっと小さいものの集合である.

**isfull(v)** : spare(v) = fosterer(v) であるとき，isfull(v) が真であると定義する．このとき，T(v) に L 分岐の条件を守ってノードの追加を行なうと必ず木の高さが増す．

isfull(v) であれば，T(v) において，すべての葉でないノードは L 個の子を持ち，葉は (T(v) での) レベル height(v) にだけ存在する．その結果 T(v) のレベル i には $L^i$個のノードが存在する．

木Tに対して以下の **P1**～ **P6** が成り立つ．

**[P1]** 木Tが性質 **P** を満たしているならば，Tの中の任意のノード v を根とする部分木も **P** を満たす． □

**[P2]** 木Tが性質 **P** を満たしているならば，Tの中の任意のノード v に対して，fosterer(v) も spare(v) もどちらも空ではない． □

**[P3]** 木Tが性質 **P** を満たしているならば，任意の v ∈ spare(R) を削除しても **P** を満たす □

(証明)v はレベル height(R) にある．v の親はレベル height(R)-1 だから，子が減っても **P** は満たされる．

**[P3']** 性質 **P** を満たしている木Tにおいて，ノードを一つ削除しても **P** を満たすのは **P3** の場合に限る． □

(証明) 葉でないノードを削除すると木が分割され,**P** が成り立たないのは自明．葉ノードは spare(R) 以外にはレベル h-1 に存在し得るが，これを削除するとレベル h-2 のノードの子の数が 1 つ減るので **P** が成り立たなくなる．

**[P4]** 木Tが性質 **P** を満たしているならば，任意の v ∈ fosterer(R) に，新たなノードを子として付加しても **P** を満たす． □

(証明)v がレベル height(R)-1 にあるならば，v の子は L 個未満であり，子を一つ追加しても **P** をみたす．さもなければ v はレベル height(R) の葉でありすべての葉でないノードは子を L 個持つ．v に子を持たせると height(R) は一つ増えるが，レベル height(R)-2 未満のノードは，ノードを増やす前の木では葉でないノードであり，すべ

て子を L 個持つので **P** は満たされる.

**[P4']** 性質 **P** を満たしている木 T において, ノードを一つ追加しても **P** を満たすのは **P4** の場合に限る. □

(証明) 子の数が L である葉でないノードに子を追加すると **P** が成り立たないのは自明. isfull(R) でないならば, レベル h-1 には子の数が L 未満のノード v が存在する. ここでレベル h のノードに子を加えると木の高さが 1 増し, v は新たな木ではレベル h-2 となるので **P** は成り立たない.

**[P5]** 性質 **P** を満たしている木 T のノード v において, isfull(v) でないならば, spare(v) ⊂ spare(R) □

(証明) isfull(v) でないので T(v) のレベル height(v) と height(v)-1 に子の数が L 未満のノードが存在する. **P** が成り立つなら, 子の数が L 未満のノードはレベル h か h-1 にしか存在しないので, T(v) でのレベル height(v) は, 木全体のレベル h に等しい.

**[P6]** 性質 **P** を満たしている木 T のノード v において, isfull(v) でないならば, fosterer(v) ⊂ fosterer(R) □

(証明)**P5** と同様.

### 4.3.5 基本戦略

**P2** より, **P** が成り立っている木には, 子を追加しても **P** が成り立つノードと取り除いても **P** を満たすような葉ノードが必ず存在する. よって以下のような基本戦略を用いる.

**S1　ノード v の停止時の処理:**

v ∈ spare(R) なら木の修復は不要. さもなければ u ∈ spare(R) を一つ選んで切り放し, v の代わりに据える.

**S2　ノード v の起動時の処理:**

任意の u ∈ $fosterer(R)$ の子として接続する. また **P5**, **P6** より, isfull(v) でない任意のノード v は, 木の他の部分にかかわりなく, 自分の子孫から起動ノードを接続す

るノードや，停止ノードの代わりに据えるノードを探して良い．

木を修復した後すぐに待機状態にならずに次の停止あるいは起動に備える処理をあらかじめ行なっておくと，木の修復を容易にすることができると考えられる．各ノードが，予め与えられている情報のほかに木を構成するのに必要な情報だけを持っているとする．この場合，例えば葉ノードが停止して，木が修復を要するかどうかを調べるだけでも $O(n)$ のメッセージを要する．よって，なんらかの付帯情報を持つことで木の維持の計算量を減らすことが妥当だと考えられる．しかし，付帯情報の更新の計算量が今度は問題になってくる．例えば全ノードに根の識別子を持たせることにすると，根のノードが停止したときには全ノードに新しい根の識別子を通知せねばならなくなる．このアルゴリズムでは，木の維持の計算量と付帯情報の更新の計算量が同程度になるように設計されている．

ノード v の持つ変数 var を v:var と表記する．v:は v が明らかな場合には省略する．また各ノードは自分自身の識別子を定数 MYID として参照するものとする．すなわち，v:MYID = v．また，特殊な識別子として NONE を使用する．これは実在のどのノードの識別子とも重ならない値とし、該当のノードがないことをあらわす．たとえば，R:parent = NONE である．

アルゴリズムでノード v が持つ変数を以下に示す．

**parent** ：親の識別子

**C(i)** ：子の識別子 $(1 \leq i \leq L)$

**NC** ：子の数

これらは木を構成するために必要な変数である．ノード間の親子関係は，ノードの持つ変数の値で定義する．

付帯情報を格納するための変数は以下の通り．

**height** ：height(v)

**isfull** : isfull(v)

**spare** : spare(v) の任意の要素.

**fosterer** : fosterer(v) の任意の要素.

**ch(i)** : height(C(i)) $(1 \leq i \leq NC)$

**cf(i)** : isfull(C(i)) $(1 \leq i \leq NC)$

**cspare(i)** : C(i):spare

**cfosterer(i)** : C(i):fosterer

**G(i)** : それぞれの子 C(i) について一つづつ，その左端の子 (v にとっては孫) の識別子.

**next** : 自分の右隣の兄弟の識別子．自分が右端の場合には祖父の識別子．根の右端の子である場合には左端の兄弟の識別子.

**rootspare** : R:spare のコピー．R の右端の子だけが持つ.

G(i), next, rootspare は，内部ノードが停止した場合に，そのノードに隣接していたノード同士が相互に連絡するための情報である (図 4.3)．このような復元に必要な情報にバランスを持たせることで，起動，停止いずれの場合も同定度の計算量になるように工夫した．height,isfull,spare,fosterer の算出手段は以下の通り.

NC=0 すなわち葉ノードの場合

height := 0

isfull := TRUE

spare := MYID

fosterer := MYID

NC > 0 の場合,

height := $max\{ch(i) \mid 1 \leq i \leq NC\} + 1$

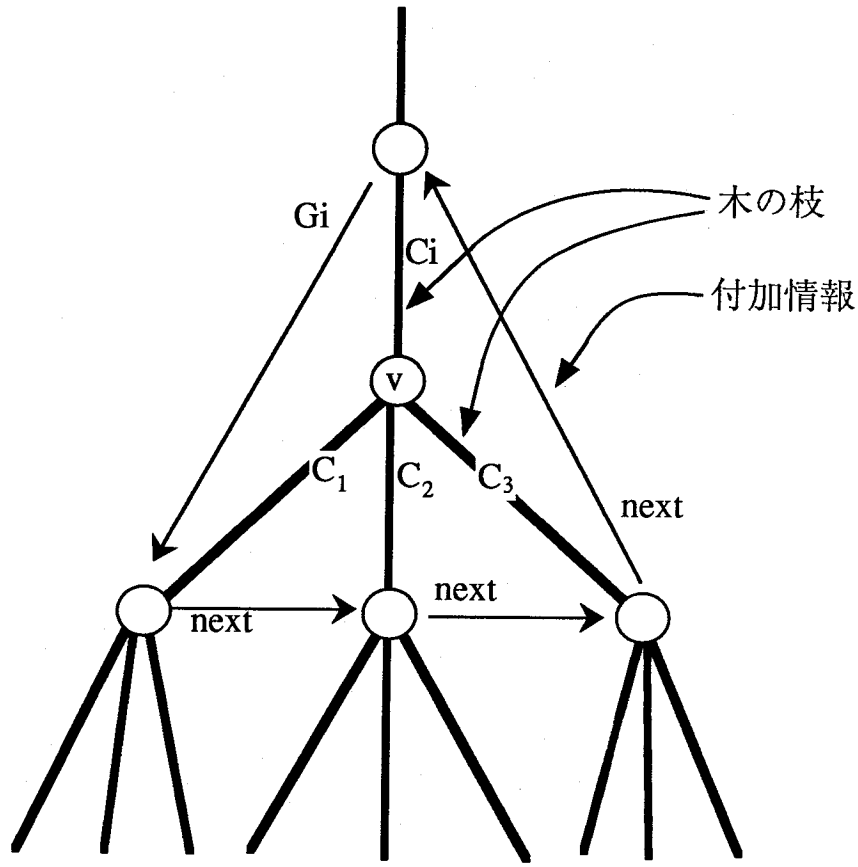



図 4.3: ノード v 停止時の連絡経路

isfull := 「NC=L かつすべての i($1 \leq i \leq NC$) について cf(i) = TRUE かつ ch(i) + 1 = height」, すなわち, 子ノードを L 個持ち, すべての子ノード i が, isfull(i) でありかつ高さが自分の高さ-1 のとき true.

fosterer := cfosterer(j); isfull ならば j は任意, さもなければ cf(j) = FALSE または ch(j) = height-2 であるような任意の j.

spare := cspare(l)

l は ch(l) = height-1 であるような任意の l.

fosterer は fosterer(MYID) の 1 要素を子ノードの fosterer の中から選んでいる. spare も spare(MYID) の 1 要素を子ノードの spare の中から選んでいる.

これらの情報は必要に応じて更新する必要がある. また, 付帯情報をこのようにして決めると, 以下の性質が成り立つ. **P** が成立している状態で考える.

**[P7]** **P** が成立しているならば, 任意のノード v において, v:fosterer ∈ fosterer(v), v:spare ∈ spare(v) である. □

(証明)

・v が葉なら height(v)=0 であり,

v:fosterer ∈ fosterer(v), v:spare ∈ spare(v) ···(∗)

・height(v) <j のとき (∗) が成り立つとする.

height(v) = j であるような v を考える. v:cfosterer(i) ∈ fosterer(C(i)), v:cspare(i) ∈ spare(C(i)) である. ch(i) = j-1 であるような i が少なくとも一つは存在する. またすべての i( ch(i)= j-1) について, spare(C(i)) ⊂ spare(v). ch(i)= j-2 または ¬ isfull(C(i)) であるようなすべての i について, fosterer(C(i)) ⊂ fosterer(v).

よって (∗) はなりたつ.

よって木の高さに関する帰納法により **P7** は成り立つ.

**[P8]** b を a の子孫とする. **P** が成立しており, かつ a:fosterer ≠ b ならば, a の任意の先祖 c において c:fosterer ≠ b. また同様にもし a:spare ≠ b ならば, a の任意の先祖 c においても c:spare ≠ b □

(証明) 各ノードの spare と fosterer は, 子ノードの spare, fosterer の中から選ばれるので明らか.

**P** と, これらの付帯情報の間には以下のような関係がある.

**[P9]** **P** が成立しているならば, R:fosterer に子を一つ追加しても **P** は成り立つ. □

(証明) **P7** より, R:fosterer ∈ fosterer(R). これと **P4** より **P9** はなりたつ.

**[P10]** **P** が成立している木のノード v において v:isfull = FALSE ならば v:fosterer に子を一つ追加しても **P** は成り立つ. □

(証明) **P7** より, v:fosterer ∈ fosterer(v). **P6** より, fosterer(v) ⊂ fosterer(R). これと **P4** より **P10** はなりたつ.

**[P11]** **P** が成立している木において, R:spare を削除しても **P** は成り立つ. □

(証明)**P7** より, R:spare ∈ spare(R). これと **P3** より **P11** はなりたつ.

**[P12]** **P** が成立している木のノード v において v:isfull = FALSE ならば, v:spare を削除しても **P** は成り立つ □

(証明)**P7** より v:spare ∈ spare(v), **P5** より spare(v) ⊂ spare(R) である. これと **P3** より **P12** はなりたつ.

**[P13]** **P** が成立している木の頂点 v において v:height + 2 = (v:parent):height ならば, v の子孫のどれを削除しても **P** は満たされなくなる. □

(証明) v の任意の子孫の木全体でのレベルは h-1 以下である. よって, どれを削除してもレベル h-2 以下のノードの子の数が L 未満になるので **P** が成り立たなくなる.

**[P14]** **P** が成立している木のノード v で,

v:isfull = TRUE,

(v:parent):isfull = FALSE,

(v:parent):height = v:height + 1

がすべて成り立ならば, v の子孫の葉の任意の一つを削除しても **P** はなりたつ □

(証明) v:isfull = TRUE より, leaf(v) = spare(v) であり, その各要素 u について level(v,u) = height(v) である. よって level(v:parent,u) = height(v) + 1 =

height(v:parent) となる． spare の定義より spare(v) ⊂ spare(v:parent)． 一方 (v:parent):isfull = FALSE. よって **P6** より, spare(v:parent) ⊂ spare(R). これと **P3** より **P14** はなりたつ.

## 4.4 アルゴリズム

待機状態では付帯情報が各ノードに正しく設定されているとする．前節で述べたように，ノードの停止時には戦略 S1，起動時には S2 に従って木の修復を行う．その後，付帯情報の再計算を行い，待機状態に戻る．

ノードが停止した場合の処理を，それが葉，根，それ以外の場合に分けてで説明する．次に内部ノードが停止した場合のノードの入れ替え手順について述べ，ノードが起動した場合の処理を示す．停止，起動いずれも最後に示す更新処理を行なう．

このアルゴリズムは [9]のアルゴリズムを改良したものである．

### 4.4.1 葉ノードが停止した場合

葉ノード v が停止した場合，v の停止を検出するのはその親ノードであり，p = v:parent が始動ノードである．

- p:spare= v ならば，v 以外の子に p:spare を変更する．
- v から始めて，u:spare ∈ spare(R) であるような v の先祖 u を探す (図 4.4(a), 4.5(a)).

isfull(w)=TRUE である v の先祖 w では，w:spare ∈ spare(R) かどうかが w 自身ではわからない．よって u は isfull(u)=FALSE である p の先祖かさもなければ根である．この時，spare を v 以外の葉に変更する処理も並行しておこなう．**P8** より，この処理は spare≠v である先祖に到達した時点で終えて良い．

v を単に削除して **P** が成り立つかそれとも木の修復が必要かは u の持つ情報で判定できる．

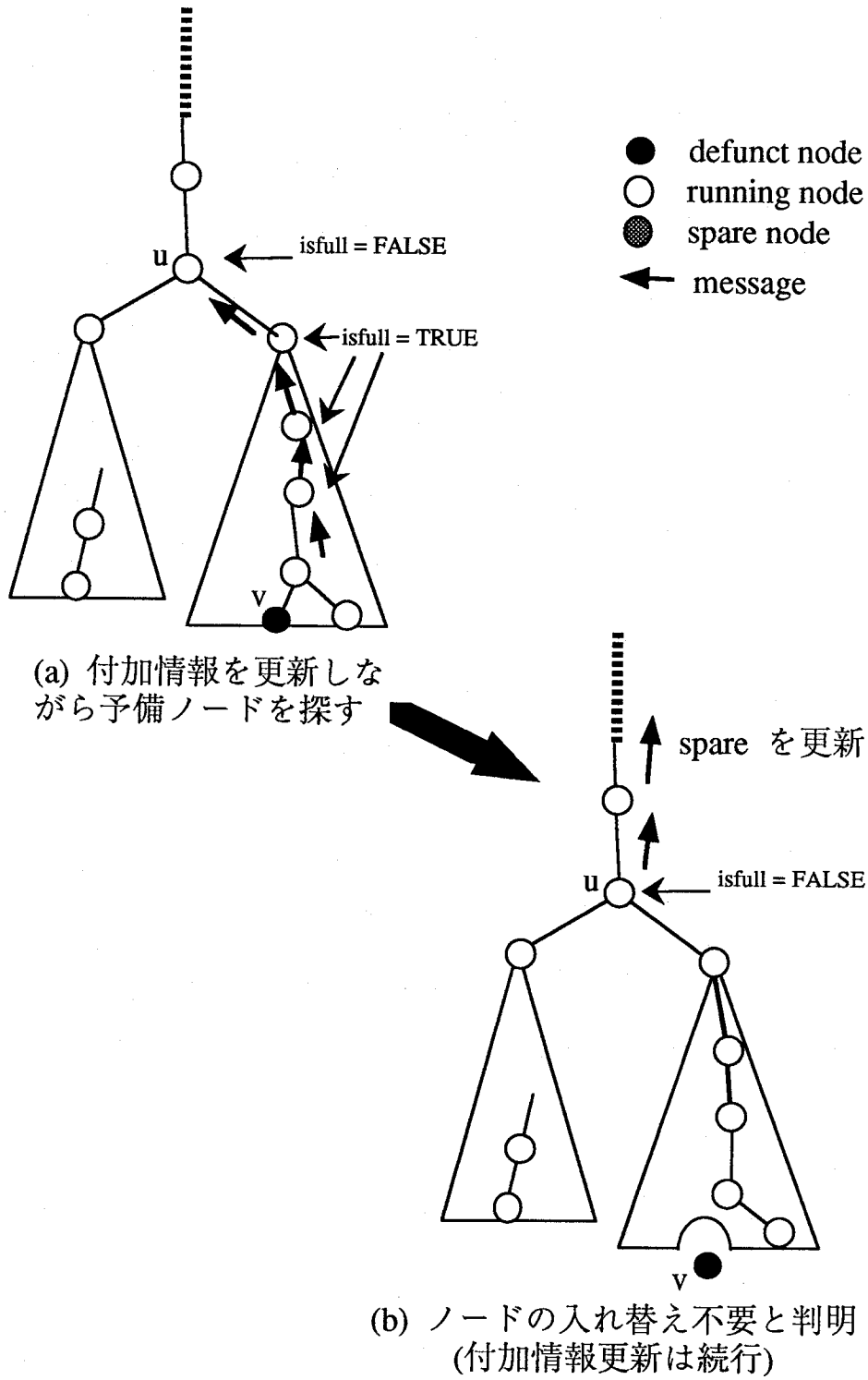



図 4.4: 葉ノード停止時の修復 (置換を不要の場合)

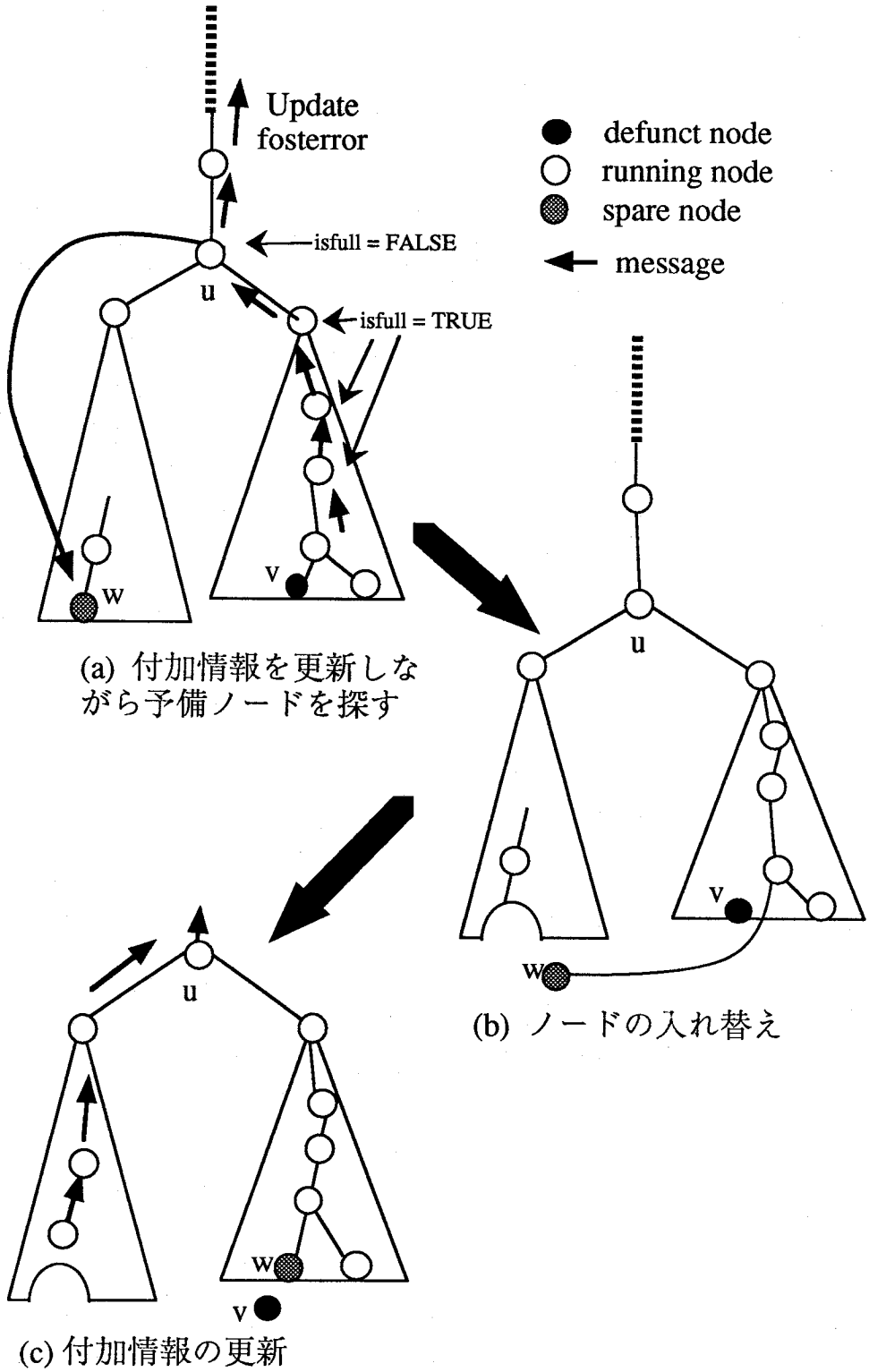



図 4.5: 葉ノード停止時の修復 (置換を要する場合)

- u:spare = v ならば，さらに先祖に向かって spare を更新してゆく．
- 木の修復が必要なら w= u:spare を v の代わりに据える (図 4.5(b))．p の spare と fosterer を再計算する時，w 以外の子から選ぶ．

### 4.4.2 根ノードが停止した場合

根ノード R が停止した場合，R に子がなかった場合は，すべてのノードが停止したことになるので何も起こらない．

R に子があれば，そのうちの 1 つ以上が始動する．始動あるいはメッセージ ⟨INFORM_LOST, parent⟩ を受信した R の子は，⟨INFORM_LOST, parent⟩ を next に 1 回だけ送る．R の右端の子 r( rootspare ≠ NONE) は始動するかあるいは INFORM_LOST の受信で起動し，r:rootspare を R の代わりに据える．

### 4.4.3 根でも葉でもないノードが停止した場合

根でも葉でもないノード v が停止した場合，

- 始動した，あるいは ⟨INFORM_LOST, v⟩を受信したノードは，next に 1 回だけ ⟨INFORM_LOST, v⟩を送る．
- v の親ノードは C(i)=v である i を求め，next ではなく G(i) に送信する (図 4.6(a))．
- v:parent から始めて，u:spare ∈ spare(R) であるような v の先祖 u を探す (図 4.6(b))．
- u:spare を v の代わりに据える (図 4.6(c))．

### 4.4.4 ノードの入れ換え処理

葉ノードの入れ換えは，その親が入れ換えを知れば良い．葉でないノードを入れ換えるには，まず各ノードの next や G(i) を用いて停止したノードの周囲のノードに

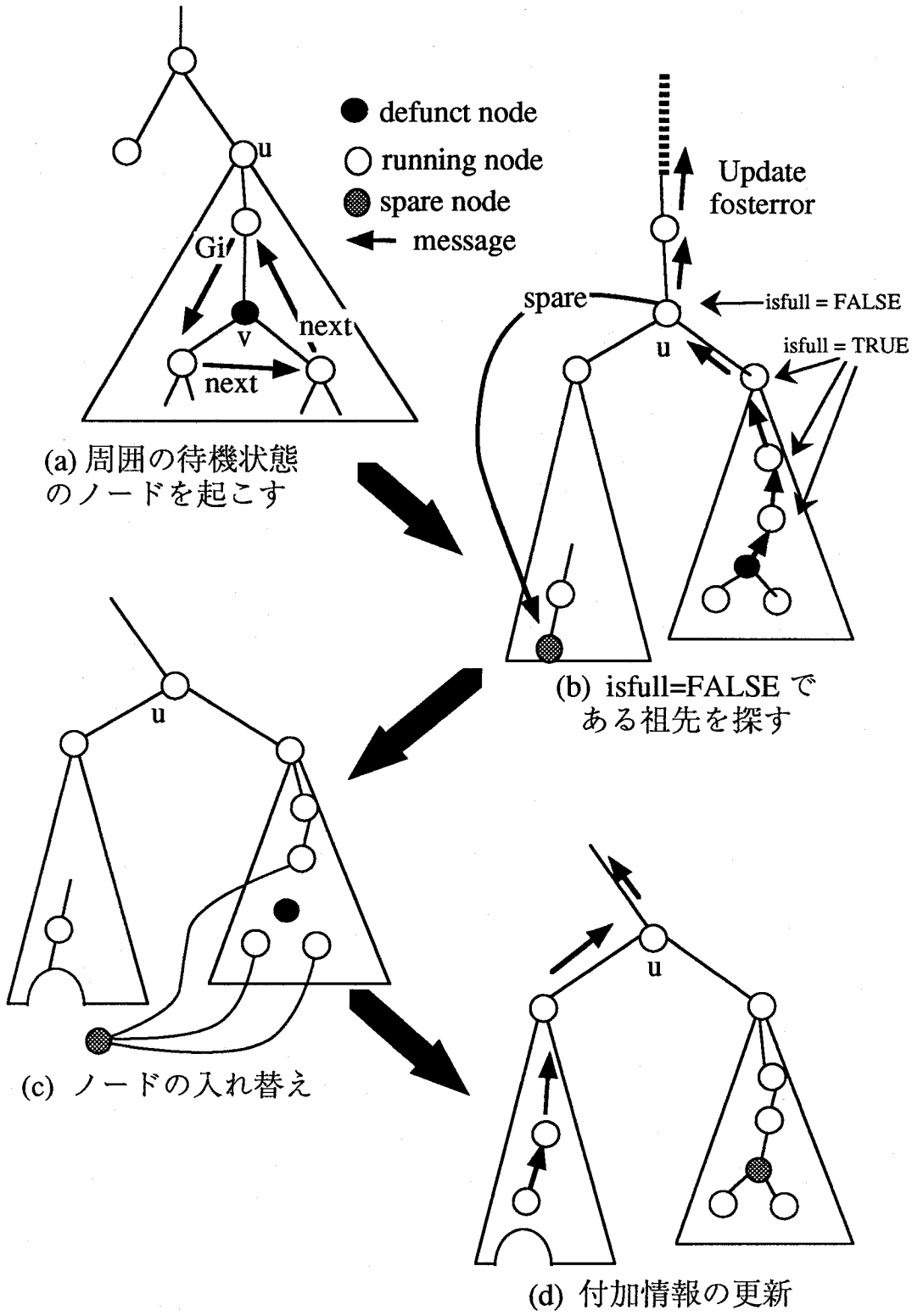



図 4.6: 内部ノード停止時の修復

代わりのノードの識別子を知らせる．隣接ノードはそれに反応し，メッセージを送って停止したノードが持っていた情報を代わりのノードに与える．また，最大 L 個の孫にも祖父の交替を連絡する必要がある．

### 4.4.5 ノードが起動した場合

ノード v が起動した場合，v は各ノードに ⟨I_WAKEUP,v⟩の送信を試みる (図 4.7(a))．すべて失敗したら，他のノードはすべて停止しているのだから v は根となる．送信が一つでも成功すれば，I_PARENT が届くのを待つ．

I_WAKEUP を受信したノードでは以下の処理を行なう．

```
if not isfull or ( parent = NONE ) then
    send(fosterer, ⟨ADDCHILD,v⟩)
else
    send(parent, ⟨I_WAKEUP,v⟩)
```

⟨ADDCHILD,v⟩を受信したノードは，I_PARENT を v に送って v を自分の子とする (図 4.7(c))．

### 4.4.6 付帯情報更新処理

4.4.1～4.4.5に述べた手順で木を修復した後に行なわれる，各ノードでの付帯情報を正しく保つ処理である．付帯情報のうちでも子ノードの高さなど他の付帯情報の算出源となる情報の変更を根に向かって伝播することによっておこなう．親に送るメッセージは ⟨UPDATE, height,isfull,spare, fosterer, c(1)⟩である．子 C(i) から UPDATE メッセージを受信したノードは，その内容を順に ch(i), cf(i), cspare(i), cfosterer(i), G(i) に入れ，自分自身の height,isfull,spare, fosterer を再計算する．変化があれば，その親に UPDATE メッセージを送る (図 4.5(c),4.6(d), 4.7(d))．ノード v の親ノードの入れ換えが起こった場合，v は v:c(NC) に新しい祖父ノードの識別子を連絡する必要がある．

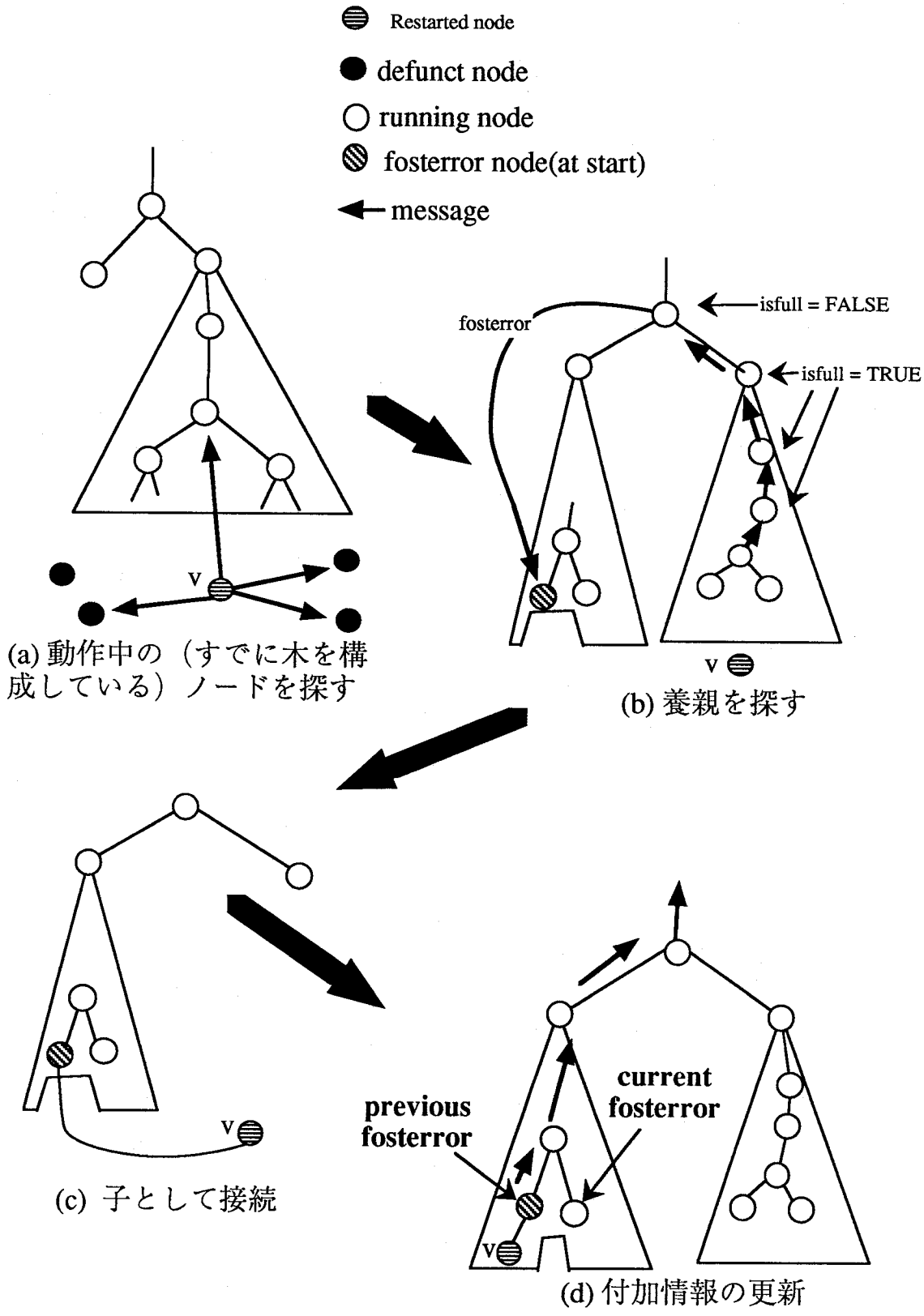



図 4.7: ノード起動時の修復

## 4.5 評価

最悪時のメッセージ数と局所計算量でアルゴリズムを評価する．通常，LAN では通信コストが小さくメッセージ数での計算量評価は不適当だと考えられる．しかし，局所計算量のうち，通信処理に要した部分の総和がメッセージ量に比例することと，多くの分散アルゴリズムがメッセージ数で評価されていることもありまず最悪時のメッセージ数を示し，次に局所計算量により評価する．

ネットワーク全体のノード数を **n**，停止中のノード数を **k** とする．木の高さ h= $\lceil \log_L((L-1)(n-k)+1) \rceil - 1$ である．

### 4.5.1 メッセージ数

それぞれの場合ごとのメッセージ数を計算する．

- ノードが停止した場合．
  - 葉ノード v が停止した場合．

    u の探索と v を spare として持つノードに spare を変更させるのに $h-2$ メッセージを要する．修復が不要だった場合には，v の削除に $O(1)$，v を削除した後の更新に $h-2$ メッセージである．修復が必要だった場合には，修復に $O(1)$，v が u:spare に入れ替わったことの更新処理に $O(1)$，u:spare を以前の親から切り離した後の更新に $h-2$ メッセージ．よって合計 $2h+O(1)$ となる．
  - 根ノード R が停止した場合．

    INFORM_LOST が $L$ 個，r:spare を R の代わりに据える処理に $3L+O(1)$，r:spare を切り離した後の更新に $h-2$ メッセージである．合計は $4L+h+O(1)$ メッセージとなる．
  - 葉でないノード f が停止した場合．

INFORM_LOST が $L+1$, u を探すのに $h-2$, u:spare を f の代わりに据える処理に $3L+O(1)$, u:spare を切り離した後の更新に $h-2$ メッセージである.

合計 $4L+2h+O(1)$ メッセージを使用する.

よって, ノードの停止時には最悪時 $4L+2h+O(1)$ メッセージを要する.

- ノード v が起動した場合.

  v が送信する⟨I_WAKEUP,v⟩が $k+1$, v 以外が送信する⟨I_WAKEUP,v⟩が $h$, u:fosterer の子にする処理に $O(1)$, 更新処理に $h-1$ メッセージを要する. よって, ノードの起動時には最悪時 $k+2h+O(1)$ メッセージを要する.

### 4.5.2 局所計算量

各ノードでのアルゴリズムの計算量を, 通常の逐次計算アルゴリズムと同様に計算する. 送受信操作の計算量は一般には $O$(メッセージサイズ) の計算量だと考えられる. しかし, ここで示したアルゴリズムではもっとも大きなメッセージでも識別子, 木の高さ, 真偽値を 6 個含むだけなので $O(1)$ の計算量だとする.

メッセージを受信してから受信待ちになるまでの計算量を考える. I_WAKEUP のような isfull や height などを参照する必要のあるメッセージの処理の計算量は $O(1)$ である.

一方, UPDATE のように各子ごとの情報を参照する必要のあるメッセージを処理するには, まずメッセージの送り元の識別子が自分のどの子のものかを調べる必要がある. 例えば配列で管理しているのなら, 識別子 C(i) から i を逆引きする必要がある. 2 分探索を用いたとして, これには $O(\log L)$ で可能である.

ある一つの子に関して, 付帯情報の更新や削除, 登録は $O(1)$ でできる. その後の更新された付帯情報からの height などの算出も $O(1)$. よって, UPDATE メッセージの処理は $O(\log L)$ である.

停止した葉でないノードの代わりとなるノードは，隣接する L+1 ノードに関する情報をすべて受信してから height などを計算する必要があるが，メッセージの送り元の識別子が自分のどの子のものか照合する必要はないので，$O(L)$ である．

ノードの停止時を考えると，停止したノードの隣接ノードをすべて起動するのには最大 h+1 ノードが関与してそれぞれの計算量は $O(1)$ である．ただし停止したノードの親ノードだけは $O(\log L)$ である．根に向かって spare を探す処理には最大 h ノードが関与して，それぞれの計算量は $O(1)$．停止したノードの入れ換え処理は，その周囲のノード (最大 L+1 個) と祖父と孫 (最大 L+1 個) がそれぞれ計算量 $O(1)$，入れ換えられるノードの計算量は $O(L)$ である．spare を切り離した後の更新処理には最大 h ノードがノードが関与してそれぞれの計算量は $O(\log L)$ となる．

よって，ノードの停止時の局所計算量の総和は $O(h + h\log L + L)$ となる．局所計算量のもっとも大きいノードは停止したノードと入れ換えられるノードで，計算量は $O(L)$ である．

ノードの起動時には，動作中のノードを探す処理に $O(k)$ を要する．根に向かって fosterer を探す処理には最大 h ノードが関与して，それぞれの計算量は $O(1)$．fosterer に接続した後の更新処理には最大 h ノードが関与してそれぞれの計算量は $O(\log L)$ である．よって，ノードの起動時の局所計算量の総和は，$O(k + h + h\log L)$．k は最悪の場合は $n-1$ なので，局所計算量のもっとも大きいノードは起動したノードで，$O(k)$ である．

# 第 5 章

# 結論

高速 LAN の性質を反映したネットワークモデルと，計算機の故障と回復双方を考慮した分散アルゴリズムについて考察した．

まず，高速 LAN を想定したネットワークモデルについて議論した．本論文で示したモデルは送信に同期した受信や，非同期に起きるノードの停止と回復など高速 LAN の特徴をモデル化しており，ネットワークの連結性の尺度についても定義している．

次に，このモデル上での情報更新問題を解くアルゴリズムを作成し，正当性を証明した．ここで示したアルゴリズムは単純ではあるが，ノードの停止，起動の状況という点から見て，これ以下ではひとつのネットワークではないという最悪の状況下で解いている．このアルゴリズムは動作条件にネットワークの状況が入っており，ネットワークの状況で正しく働くかどうかが決まる．またネットワークの状況が変化しても，実行の所要時間はほとんど変わらない．ネットワークの状況の変化につれてアルゴリズム実行の所要時間が増えるかわりにより悪いネットワークの状況でも動作するアルゴリズムの研究は今後の課題である．

第 3 に，ネットワーク上での階層的なサーバクライアント関係を，計算機の停止や起動に対して，バランスのとれた構造として維持することをモデル化した問題である完全 L 分木維持問題について考察した．完全 L 分木は，一つのノードが持つことができる子 (一つのサーバが直接サービスできるクライアント) の数を限定した場合に，

各ノードから根(マスターサーバ)までの間の階層数を最小にする構造である．n 個のノードのうち k 個が停止している時，階層数が $h = \lceil \log_L((L-1)(n-k)+1) \rceil - 1$ となる．ここで示したアルゴリズムは，ノードの停止時にメッセージ数 $4L+2h+O(1)$，局所計算量の合計 $O(h+h\log L+L)$ で，起動時にはメッセージ数 $k+2h+O(1)$，局所計算量の合計 $O(k+h+h\log L)$ で完全 L 分木の維持を行う．

分散アルゴリズムの評価には通信計算量(メッセージ数)や理想時間計算量が広く用いられている．これらの尺度はメッセージあたりのコストやメッセージの遅延時間がアルゴリズムの実行のコストや計算時間の主な要因であるような場合にうまく適合する．しかし，高速 LAN のように低コストで遅延時間の短い状況でのアルゴリズムを評価するためには，ノード内部での計算量が無視されてしまうため不適当である．第4章でアルゴリズムの評価に用いた局所計算量は各ノードでの計算負荷を表すものである．この尺度では修復が完了するまでの理想時間計算量が同じであっても，送受するメッセージ数が少なかったり，ノード内でのメッセージを処理する効率が高ければ計算量の評価は小さくなる．

今後の課題としては，現在置いているノードの停止や起動が一つづつ起きるという仮定をなくしても動作するアルゴリズムの作成や，それぞれ問題の計算量の下界を求めることがある．特に局所計算量については，データ構造などを工夫することで改善の余地がある．

局所計算量は高速 LAN を想定したネットワークモデル上での分散アルゴリズムの計算負荷に対応する尺度であった．同様に分散アルゴリズムの実行の経過時間の尺度を考案することも必要である．

本論文では 1 対 1 のメッセージ交換だけを用いてモデルを構成したが，実際のネットワークにおける通信形態は，同報通信や回線交換型の通信など多様である．これらに対応するようにモデルを拡張することも興味深い問題である．

現在ネットワークの進歩はめざましく，高速化，大規模化，大容量化が進められている．そのため本論文で LAN の属性としてとりあげた高速性，均一性といった特性

が，さらに大規模なネットワークでも提供されるようになりつつあり，本論文の結果の有用性が高まっている．またネットワークの進歩にともないプロトコルの標準化が進められ，提供される機能の抽象度が高くなってきている．たとえば，point-to-point の通信線を使用している場合でも，個々の通信リンクを指定したメッセージ送受信操作はユーザプログラムには解放されない．ネットワークに対応したアプリケーションプログラムは今後数多く作成されると考えられ，そのアルゴリズムの基礎となる分散アルゴリズムの重要性は高い．本論文では高速 LAN を対象としたが，その他の広域ネットワーク等に対しても，アプリケーションプログラムに提供されるネットワークのプログラミングモデルに適合したネットワークモデルの作成やその上での分散アルゴリズムの研究を行うことが必要である．

# 謝辞

本研究の全過程を通じて，直接御理解ある御指導を賜わり，常に励ましていただいた都倉信樹教授に深く感謝の意を表します．

大学院前期および後期課程において御教示，御指導いただいた情報工学科の嵩忠雄教授，橋本昭洋教授，菊野亨教授，欅田栄一教授，首藤勝教授，鳥居宏次教授，脇田壽教授，谷口健一教授，宮原秀夫教授，谷内田正彦教授，柏原敏伸教授，教養部の藤井護教授，大阪大学産業科学研究所の豊田順一教授，北橋忠宏教授，溝口理一郎教授，医学部の田村進教授，並びに角所収名誉教授，故藤澤俊男教授，故高嶋堅助教授に心から深謝致します．

大学院を通じて御指導いただいた西谷紘一助教授，伊藤実助教授，David Notkin 助教授，情報処理教育センターの西尾章治郎助教授に深く感謝致します．

本研究を通じて有益な御討論，御指導いただいた萩原兼一助教授に心から感謝いたします．

本研究の全過程を通じて，有益な御助言，御指導いただいた辻野嘉宏講師に厚く御礼申し上げます．

大学院を通じて有益な御指導，御援助いただいた井上克郎講師，東野輝男講師，増田澄男講師，藤原融講師，下條真司講師，関浩之講師に深謝致します．

種々の面で御助言，御援助いただいた工学部情報システム工学科の荒木俊郎助教授，情報工学科の増澤利光助手，魚井宏高助手に心から感謝いたします．

さらに，著者の在学中，御討論いただいた都倉研究室の方々，とくに，都志武史氏(現クボタコンピュータ株式会社)に感謝いたします．

# 文献


(1) 上谷晃弘: "ローカルエリアネットワーク — イーサネット概説 —", 丸善株式会社 (1985).

(2) 萩原兼一: "分散アルゴリズムの複雑度について", *Proceedings of The Logic Programming Conference '87*, pp.11-20(June 1987).

(3) P. Weiss: "Yellow Page Preotocol Specification", Sun microsystems, Inc.(1985).

(4) Leslie Lamport: "Time, clocks, and the ordering of events in a distributed system", *Communications of the ACM*, **21**, 7, pp.558-565(July 1978).

(5) Maekawa,M., Oldehoeft,A., Oldhoeft,R.: "Operating Systems", pp.204-206, The Benjamin/Cummings Publishing Company, Menlo Park, Calfornia(1987).

(6) 齊藤, 都志, 辻野, 都倉: "高速 LAN のモデルとその上での更新アルゴリズム", 信学技報, **COMP88**-14, (1988-05).

(7) 齊藤, 都志, 辻野, 都倉: "高速 LAN のモデルとその上での更新アルゴリズムについて", 信学論 (D I) , **J72-D-I**, 2, pp.108-116(1989-02).

(8) Riccardo Gusella, Stefano Zatti, and James M. Bloom: "The Berkeley UNIX Time Synchronization Protocol", UNIX System Manager's Manual, 4.3BSD, Virtual VAX-11 Version, University of California, Berkeley, California(1986).

(9) 齊藤, 辻野, 都倉: " 高速 LAN モデルにおける生成木維持問題", 信学技報, **COMP89**-22(1989-06).

(10) D.Stott Parker,Jr. and Behrokh Samadi : "Adaptive distributed minimal spanning tree algorithms", *Proc.IEEE Sympo. on Reliability in Distributed Software and Database Systems*, pp.138-144(1981).