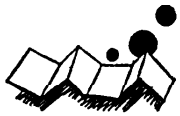
解　説

2. 基　礎　技　術



2.3 分散処理の高信頼性技術†

長谷川　聡††　阪田史郎††

1. はじめに

コンピュータを利用した各種ネットワークシステムが社会のあらゆる分野に浸透しつつある現在，その障害や誤動作による社会的影響，経済的損失は計り知れないほど大きなものになる可能性がある．分散処理システムの高信頼化は，時代の要請となっている．

高信頼化技術については，従来よりさまざまな概念，方式が提案されているが，現在実用に供している部分のほとんどは集中処理システムを対象としたものである．分散処理システムを対象とした方式も，70年代後半より急速に研究が進展しているが，通信遅延の問題，同期制御，障害の波及など分散化による処理の複雑さに起因する問題から，実システムへの適用はこれからという状況である．本稿では，今後の応用が期待される分散処理システムを対象とした高信頼化技術に関し，各手法の位置付け，狙い，現状及び将来動向を述べる．

一般に高信頼化においては，対象システムやその目的，すなわち障害の防止，検出，除去（復旧，以下リカバリ）のどのフェーズに対応するかにより，適用する手法が異なる．手法自体もさまざまな観点からの分類が可能であるが，特にリカバリに対しては，対照的かつ相補的なアプローチとして表-1 に示すように，前向き回復処理（Forward Error Recovery）と後ろ向き回復処理（Backward Error Recovery）に大きく分類できる[1),2)]．この分類は，特に分散処理システムを想定したものではなく，ソフトウェアの高信頼化に関連して従来議論されてきたものであるが，各手法は今後基本的な考え方として広く分散処理にも適用されていくと思われる．

† Reliability Techniques in Distributed Computing Systems by Satoshi HASEGAWA and Shiro SAKATA (C&C Systems Research Laboratories, NEC Corporation).

†† 日本電気(株)C&C システム研究所

さらに，手法の性質に着目した分類として，最も基本的な技術としての冗長化，障害の波及を最小におさえ矛盾なく処理を再開するための同期制御に重点をおいた構造化と独立化，プロセスやネットワークの分散アルゴリズムによる再構成，障害の検出，回避を目的とする監視，などを考えることもできる．しかしこれらは互いに密接に関連し，必ずしも明確に区別できるものではなく，組み合わせで用いられるのが一般的である．

一方，分散処理システムにおけるリカバリ手法の適用対象は，大きくプロセスまたはプロセッサ，ネットワーク，データベースに分けられる．プロセスについては，密結合分散システムに対してソフトウェア工学に基づく各種手法が古くから提案されている．ネットワークについては，サイトとリンクの結合状態の回復を主目的とする．データベースについては，管理されるデータの保全を目的とし，同時更新制御（Concurrency Control）技術がその基本となっている[3),4)]．以下では，高信頼化においてその適用効果の大きいリカバリを目的とした技術，特に基本的かつ汎用的な技術として重要な位置を占める，構造化・独立化，及び冗長化の手法を用いた代表的な技術について述べる．

2. 高信頼化のための構造化・独立化技術

構造化と独立化は，本来リカバリ処理のみを対象としたものではなく，主にプログラム開発の容易性，システムの保守性，拡張性の向上を目的とした概念である．リカバリに対する構造化と独立化の目的は，障害の早期発見，障害の波及範囲の局所化・限定化，復旧対象部分の明確化である．構造化とは，システムの設計時にあらかじめこれらの目的を達成するために構造を設定することであり，独立化とは構造化された部分の間での独立性を保証しながら，同期制御を含め効率的にシステムを実行させることである．

表-1 リカバリ処理の分類

処理種別	内 容	方式例
前向き回復処理	障害発生したシステムの状況と想定される障害原因の分析に基づき前進的に回復する	例外処理 (exception handling) 補償 (compensation)
後ろ向き回復処理	障害発生以前の状態を保存しておき障害検出時にその状態に戻すことにより回復する	リカバリブロック，履歴情報によるロールバック，コミットメント制御

ここでは，構造化と独立化の最も基本的な枠組みであるアトミックアクション (Atomic action) について概説し，その分散処理システム，特に分散データベースへの応用として代表的なコミットメント (Commitment) 制御について述べる．

2.1 アトミックアクション

リカバリを行うためには，障害発生時にシステム内のプロセスがどこまでを正常に実行したかを知る必要がある．そのためプロセスが実行完了したかまたは全く実行していないか (all or nothing property)，また正常完了したか否かが判定できるような処理単位の設定が必要となる．このような単位として導入された概念がアトミックアクションであり，データベースにおけるトランザクションを一般化した概念である．Lomet は従来のセマフォ，臨界領域 (critical region) によるプロセス間の同期制御のメカニズムを形式的に記述するため[6]，Lampson & Sturgis は次節で述べる2フェーズコミットメントを説明するための手段としてアトミックアクションの概念を用いている[5]．

さらに，アトミックアクションを障害波及範囲の限定化の観点から一般化した，SOC (Sphere of Control) の概念が Davies[7]，Bjork[8] により提案されている．この SOC は，Randell[1] の提唱するリカバリラインと類似した考え方に基づいている．以下ではアトミックアクションを適用したリカバリラインを説明する．

一般に，システム内のすべてのプロセスの動作に対してアトミックアクションを適用することは，現実的に不可能である．また複数のプロセスが相互に関係しながら処理を実行する場合には，各プロセスごとにアトミックアクションの実行開始点（リカバリポイント）に戻るだけではリカバリができず，システム全体の再実行が必要となるいわゆるドミノ効果 (図-1) を引き起こす危険性がある．ドミノ効果を防ぐ手法として，Randell はリカバリラインの探索方法を提案している．リカバリラインとは，システム全体としてプロセス間の相互作用の排除が可能なリカバリポイントの組である．図-1 のドミノ効果に対しては図-2 のようなリカバリラインが考えられる．しかし現実にはリカバリラインを考慮せずに各プロセスが処理を実行した場合，リカバリラインの探索が困難になるため，あらかじめシステム設計時に設定しておく方法が一般的である．

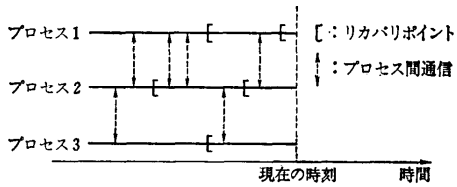


図-1 ドミノ効果

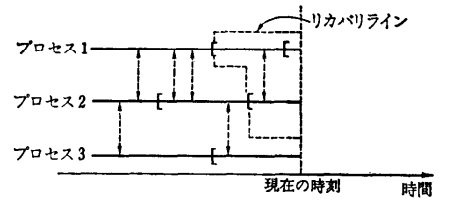


図-2 リカバリライン

最近では，CSP (Communicating Sequential Processes)[9] や前向き回復処理制御[10]，共有データへのアクセスに関する同期制御と共有データの回復処理とを統一的に扱う効率的な分散データベース制御[11]，さらには耐故障性を維持するプログラミング言語[12]に対するアトミックアクションの適用，ならびに実システム上での実証検討がなされている．

2.2 コミットメント制御

コミットメント制御は70年代半ばのオンラインデータベースの出現とともに重要視されてきた概念であり，その後の分散データベースへの適用を想定し，OSI (Open Systems Interconnection)[13] や DCNA (Data Communication Network Architecture)[4] などのネットワークアーキテクチャにも取り入れられている．リカバリの観点からはプロセス間の同期の乱れが生じないよう，または同期の乱れが生じても回復させるような同期制御の代表的なメカニズムとして位置づけられる．

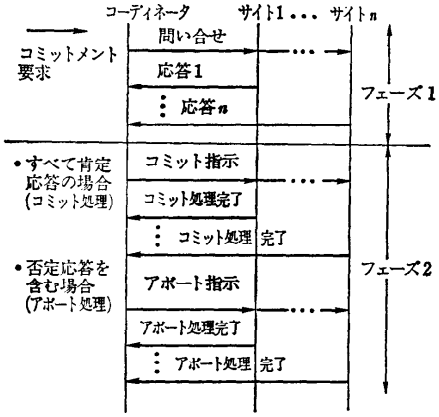


図-3 集中型2フェーズコミットメント

2フェーズコミットメントは，Lampson & Sturgis, Lindsay[14] らにより提案された方式で，二つのフェーズを同期制御中に設定し，関連する複数の分散プロセスが，プロセス間の同期制御情報を消失したり，プロセスの存在するサイトがダウンした場合でも同一の決定を行い，処理の完全性を保持することを目的としている．代表的な方式は図-3 に示すような集中型2フェーズコミットメントである[5]．

第一フェーズ…関連する全サイトが統一的にコミット可能か否かを調べるフェーズであり，可否を問われた各サイトは応答を返すと同時に利用中の資源をロックし，第二フェーズまで制御情報を保持し続ける．

第二フェーズ…関連する全サイトからコミットできるか否かの応答を受け，すべて肯定応答ならコミット，否定応答が含まれるならアボートの指示をコーディネータが出し，全サイトが同一の決定を行う．

2フェーズコミットメントは全体としてアトミックアクションを実現するものである．さらに，制御メッセージの転送順序をあらかじめ決めておくことにより，コーディネータを必要としない分散制御型の方式や集中型と分散型を組み合わせた階層型[15]なども提案されている．いずれにしても，2フェーズコミットメントは分散処理環境における処理結果の完全性の保証を与えるものであり，現実のシステムに適用する際は，サイトあるいはデータの分散状況に応じて，採用する方式の選択，通信オーバヘッドを減少させる工夫などが必要である．

3. 高信頼化のための冗長化技術

分散システムにおいては，その構成要素は多様かつ複雑であり，各構成要素で発生し得る故障もまた多様なものとなる．このような状況のもとで，あらゆる故障 (fault) を想定して回復処理を用意することは，一般に困難であり，回復不可能な故障の起こる確率も無視できない．これらの理由から，システムに冗長性を持たせることは高信頼化において本質的に重要である．本章では，冗長性により高信頼度を実現するための，代表的な基礎技術について，プロセスの冗長化及び通信リンクの冗長化の観点から述べる．

プロセスの冗長化に対しては，投票アルゴリズム，主にマイクロプロセッサなどの密結合分散システムへの適用が期待されるビザンチン合意 (Byzantine Agreement) アルゴリズムを取り上げた．多数決論理は冗長化によるリカバリ技術の基礎であり，ビザンチン合意を行うための核となる．一方，通信リンクの冗長化に対しては，コンピュータネットワークやデータベースなどの疎結合分散システムを主対象とした高信頼度同報通信，高信頼度ルーティングを取り上げた．これらの技術は，冗長プロセスによる高信頼化を実現するための，通信ネットワークに課せられた基礎技術である．

3.1 投票アルゴリズム

分散システムにおいて，故障が生起した後も正常プロセスを再構成し，処理を継続することが必要な場合が多い．この場合再構成の指揮をとるプロセスを選出する必要がある．通信リンク障害によりネットワークが分断されない場合には，唯一の指揮プロセスを選出すればよく，いわゆる相互排除 (mutual exclusion) と等価な問題となる．たとえば，Garcia-Molina はプロセスにあらかじめ番号を割り当てておき，正常プロセスの中で最大番号を有するものを指揮プロセスとし，相互排除を実現している[16]．しかし，通信リンク障害によりネットワークが分断される場合には状況は複雑化し，複数の指揮プロセスが管理下のグループプロセスを制御するようにせねばならない．通信リンク障害が復旧し，グループプロセスの構成が動的に変化し得ることを考え，勧誘 (invitation) に基づくアルゴリズムが考えられている[16]．これは，各指揮プロセスが周期的に他のグループプロセスを自グループに勧誘し，受理されればグループを再構成していく手法である．

通信障害によりネットワークが分断される場合には，別の問題点も生じる．たとえば，分散データベースの管理を考えてみる．ネットワーク分断により独立した複数のプロセスグループが生成され得るが，デー

タベースの更新を各グループが独立に行うと，分散データの散逸を生じる可能性がある．これを避けるためには，相互排除，すなわち唯一のプロセスグループだけがデータベースの更新管理を行うメカニズムが，通信障害発生時にも実現される必要がある．Garcia-Molina などはこの問題に対して，多数決に基づく方法の基礎検討を加えている[17]．これは，各プロセスにあらかじめポイントを割り当てておき，グループ内の総ポイント数が全システムの総ポイント数の過半数を超えるグループだけがデータベースの更新管理の権利を得るものである．今，a, b, c, d の4プロセスでシステムが構成されているものとしよう．単純に各プロセスに1ポイントずつ割り当てるものとすると，データベースの更新権を得るには3ポイントを有する必要がある．これは次に示すグループ集合 S と等価である．

$S=[\{a,b,c\}, \{a,b,d\}, \{a,c,d\}, \{b,c,d\}]$

一方プロセス a に2ポイント与え，他プロセスに1ポイントずつ割り当てるものとすると，やはりデータベース更新権を得るには3ポイント必要ではあるが，対応するグループ集合は以下の R のようになる．

$R=[\{a,b\}, \{a,c\}, \{a,d\}, \{b,c,d\}]$

後者のポイント割り当ての方が，前者のそれよりも耐故障性の観点からは明らかに優れている．これは，S のもとで動作するプロセス集合が R のもとで動作するプロセス集合に完全に含まれるからである．たとえば，通信障害によりシステムが $\{a,b\}$ と $\{c,d\}$ に分断された場合，R のもとでは一つのデータベース更新可能なプロセスグループが存在し得るが，S のもとでは存在しないことからもわかる．性質のよいポイント割り当て方法の詳細は，文献[17]に譲ることにする．

3.2 ビザンチン合意

システムの構成要素の故障モードは通常複雑であり，単純な故障を仮定することは現実的でない場合が多い．よって，故障に関するいっさいの仮定を設けない考え方がある．このような任意故障のもとで，各プロセスがメッセージを交換することにより目的変数値の“合意 (agreement)”をとることをビザンチン合意と呼び，次の条件が成立する場合に成り立つ[18),19)]．

（1） すべての正常プロセスは同じ値に合意する．

（2） メッセージ送信プロセスが正常ならば，すべての正常プロセスはその送出された値に合意する．

簡単な例を用いてビザンチン合意を説明する[18)]．

いま，n 人の“指令官”がおり，1人の“指揮官”と $(n-1)$ 人の“副官”で構成されているとする．m 人の“反逆指令官”が存在し得る環境下において，指揮官と副官の間で攻撃命令の合意をとる例を考えてみる．ここで合意がとられるべき目的変数 v は，“攻撃”，“退却”の2値をとるものとする．$n \geqq 3m+1$ の場合に，次に示す再帰アルゴリズム OM(i) にてビザンチン合意がとられる．

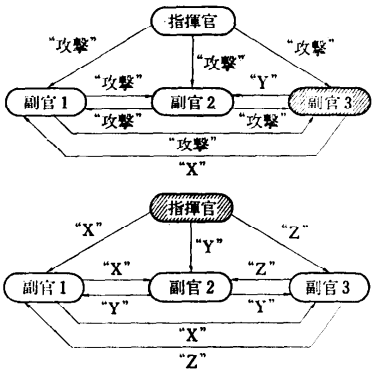


図-4 ビザンチン合意例

アルゴリズム OM(0)：

STEP 1： 指揮官は変数 v をすべての副官に送出

STEP 2： 副官 i は指揮官からの値 v に合意する

アルゴリズム OM(m)，$m>0$：

STEP 1： 指揮官は変数 v をすべての副官に送出

STEP 2： 副官 i での変数値を v_i とすると，$v_i=v$ として指揮官であるかのごとく OM$(m-1)$ を実行

STEP 3： 副官 i では全受信系列 $(v_1, v_2, \cdots, v_{n-1})$ の多数を占める値を新たな v_i とする

ここで，OM(m) は m 人の反逆指令官がいても合意を保証するアルゴリズムである．もちろん，$n \geqq 3m+1$ は満足せねばならない．また，指揮官からのメッセージがないことは検出可能であるものとする．図-4 に示す例で副官3が反逆指令官であるとすると，副官3がいかなる値を送出しようとも，OM(1) 実行後に副官 1，2 は，

$v_1=$majority (“攻撃”, “攻撃”, X)＝“攻撃”

$v_2=$majority (“攻撃”, “攻撃”, Y)＝“攻撃”

となり，指揮官が始めに送出した値に合意する．また，指揮官が反逆指令官であるとすると，指揮官がいかなる値を送出しようとも最終的な各副官の値は，

$v_1=$majority (X, Y, Z)

$v_2=$majority (X, Y, Z)

$v_3=$majority (X, Y, Z)

となり，同じ値に合意することがわかる．厳密なアルゴリズムの正当性の証明は文献[18]に譲る．

以上で示した基本的アルゴリズムが収束するには，$(m+1)$ のフェーズ数と $0(n^{m+1})$ のメッセージ交換数が必要であり，n が大きいと現実的ではなくなる．このフェーズ数と交換メッセージ数，すなわち合意に達するまでの時間と必要な通信情報量の間には，トレードオフの関係があることが知られている[20]．ビザンチン合意の研究の一つの方向として，収束時間と交換メッセージ数を考慮し，より高速な，あるいはよりメッセージ量の少ないアルゴリズムが検討されている．

ビザンチン合意の研究の別の方向として，システムが同期か非同期かによるアルゴリズム検討がある．先程の例ではフェーズの概念があったが，これは暗にシステムに共通のものを示し，各プロセスで同時刻に始まり同時刻に終了し，常に各プロセスが同一フェーズ値を有することが仮定されている．このようなシステムモデルを同期システムと呼ぶ．一方，非同期システムではシステム共通フェーズが存在せず，非常に処理の遅いプロセスも存在し得る．このような非同期システムに対する合意アルゴリズムとして，M. O. Rabin はランダムビザンチン合意を提案している[21]．基本的には，合意値を決定するアルゴリズムを，各プロセスの局所フェーズごとにランダムに選択する方法である．

図-5 にプロセス G_i でのランダムビザンチン合意アルゴリズムを示す．総プロセス数を n，最大故障プロセス数を m としている．各プロセスでは，#k ステージにてまず他のプロセスの変数値を集める．ここで，故障プロセスはメッセージを送出しないこともあり（非同期システムではこれを検出できない），すべてのプロセスからのメッセージを待つとデッドロックに陥る可能性がある．よって，$(n-m)$ 個のプロセスからのメッセージを待つ．集められた変数値の中の多数を占める値を temp(i) に，その変数値を送信してきたプロセス数を count (i) にそれぞれ格納する．各メッセージには，改ざんを避けるため認証が必要である．次に，#k ステージでの2値ランダム値を A. Shamir のアルゴリズムに従い決定する．全正常プロセスは Lottery の手続き完了後は同一のランダム値 s を有することが保証される．続いて #k ステージでのディシジョンアルゴリズムをランダム変数 s にしたがい選択し，変数値を決定する．この方法によると，系の状態にかかわらず s が0か1かのどちらか一方の場合に合

```
Procedure BAP; (for Gi)
begin
    message ( i ) := Mi; (Gi の初期値)
    For k=1 to R do (k は Gi に固有のステージ値)
    begin
        Polling;
        Lottery;
        Decision;
    End;
End;    (BAP)
Procedure Polling;
begin
    send σi (message( i ), k) to all; (σi は Gi の認証関数)
    collect incoming σi (message ( j ), k);
            (n-m 個の値を受信するまで)
    temp( i ) := M; (M=majority ((message( j ), k)) )
    count( i ) := |{ j; (message ( j ), k)=temp( i )}|;
            ( (message( j ), k)=M のプロセス数 )
end;        ( Polling )
Procedure Lottery;
begin
    ask for Ej^k from all;
    send Ei^k to all; (Ej^k はプロセス j がステージ k で有して
                        いるランダム値)
    wait until m different values of Ej^k have arrived;
    compute sk from received Ej^ks and Ei^k;
      (A. Shamir のアルゴリズムより，正常プロセスは同じラ
      ンダム値 s を得る)
end;  (Lottery)
Procedure Decision;
begin
    s := sk; (sk は #k ステージでのプロセス Gi のランダム値)
    If(s=0 and n/2≦count( i )) or (s=1 and n−2m≦count( i ))
    then
        message := temp( i );
    else
        message := "system fault";
    end:  (Decision)
```

図-5 ランダムビザンチン合意アルゴリズム

意し，s が確率 1/2 で1と0をとるものとすると，R 回のステージの繰り返しで $1-2^{-R}$ の確率で合意に達することになる．さらに，各プロセスが合意に達したと判断すると全プロセスに通知することで，合意エラーのないアルゴリズムも可能である．この場合は合意に達するまでのステージ数は一定にはならず期待値で与えられる．

ビザンチン合意アルゴリズムは，1980年に L. Lamport らにより発表されて[19]以来，米国を中心として活発な検討がなされている．しかし，いまだアルゴリズムに要求されるオーバヘッドは大きく，実用に供するにはさらに時間がかかることが予想される．今後は，障害に対する仮定の緩和，あるいはランダムビザンチン合意にみられるような合意エラーを許容して効

率を高める，実用に近いフェーズの研究がより要求されよう．

3.3 高信頼度同報通信

同報通信は，各種分散処理アプリケーションを実現するための基本的な通信プリミティブの一つであり，これを信頼性高く実現することは重要である．現実に，分散アルゴリズムには高信頼度同報通信を暗に仮定しているケースが多い．このような背景から，サイトあるいは通信リンク障害などによるメッセージ消失に対し，正常サイト集合が同報メッセージを同じ順序で，正常受信することを保証するのが，技術の目的である．

一斉同報媒体での同報通信に対し，J. M. Chang らは巡回トークンの概念を用いたプロトコルを提案している[22]．メッセージはすべての受信サイトに同報され，トークンサイトと呼ばれる一つの受信サイトがメッセージに対してシーケンス番号を打刻し，ACK を同報する．シーケンス番号が個々のメッセージに対し唯一のサイトで管理されるので，受信メッセージの順序制御が容易に実現される．各受信サイトでは，予期されたシーケンス番号とは異なるメッセージに対する ACK を検出すると，ACK を送出したトークンサイトに対して再送を要求する．耐故障性を高めるため，トークンサイトは ACK が送出されるごとにトークン巡回リストに従い受け渡されていき，全受信サイトを巡回する．デッドロックを避けるために，ある一定期間同報メッセージがこないと NULL メッセージに対する ACK を送出することでトークン所有権を移動させている．この手法によると，あるトークンサイトがトークン所有権を離し，再びトークン所有権を得たときに，前回トークンを有していたときに ACK を送出したメッセージがすべてのサイトに受信されたことがわかり，そのメッセージをバッファから解放することができる．制御メッセージの流れを図-6 に示す．このプロトコルは同報メッセージ源とトークンサイトの間は ACK メカニズム，トークンサイトと同報受信サイトの間は NACK メカニズムに基づいており，制御メッセージ数を著しく減少させることが可能となる．さらにサイト故障に対しては，投票アルゴリズム，多数決論理に従いトークン巡回リストの更新，新しいトークンサイトの選出を行い，同報受信サイトを再構成して通信の継続性を保っている．

以上のプロトコルは，基本的には空間的な集中化の制約条件により高信頼度，順序制御が実現されている．別のアプローチとして，同報メッセージの送出時刻に制約条件を与え高信頼性を得ることも可能である．L. Lamport はシステムに絶対時刻を与え，周期的な絶対時刻タイミング時にのみ同報メッセージを送出できるメカニズムを提案しており[23]，これに基づき F. Cristian などはアトミック同報の概念のもとに高信頼度同報通信を実現している[24]．さらには，空間的集中化，時間に対する制約のハイブリッド型もある[25]．

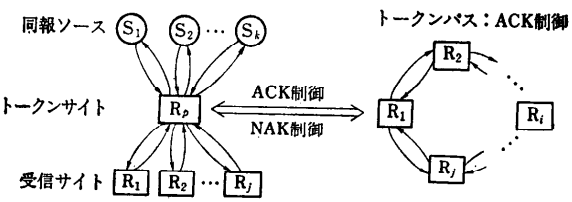


図-6 高信頼度同報プロトコル

一方，point-to-point ネットワークにて高信頼度同報通信を実現するためには，同報ルーティング制御が要求される．一方法として，全受信サイトが同報メッセージを受信すると，それを入力サイトを除くすべての隣接サイトに転送する，フラッディング(flooding)[26]がある．この手法は，ネットワークの分断がない限り通信は成立し信頼性は高いが，無効メッセージ転送が増大するとともにメッセージの無限ループが生成されるので，ループカウンタにより回避する必要がある．

フラッディングのリンク使用効率の悪さを改善する一手法として，最尤パスをメッセージ受信側で逆に求め，所望の隣接サイトからきたメッセージだけを全隣接サイトに転送する逆パス探査方法 (reverse path forwarding) がある[27]．所望の隣接サイト以外からのメッセージは棄却されるので，メッセージの無限ループは回避できる．この手法はルーティングテーブルを必要とするが，リンクの使用効率は上がる．しかしながら，ルーティングテーブルを動的に更新していく場合には，メッセージの二重受信，消失が起こる可能性があり，信頼性の点からはフラッディングに劣る．この問題点に関し，A. Segall などは種々の制御メッセージを隣接サイト間でやりとりすることで解決しているが[28]，制御の複雑さ，過大なオーバヘッドの点で難がある．なお，ルーティングテーブルの動的な更新に対しては，W. D. Tajibnapis が完全分散型の比較的エレガントな解を与えている[29]．

以上，同報通信に関しては基本方式自体の研究もいまだ数少なく，信頼性をも考慮したプロトコルは今後

の大きな研究課題の一つであると思われる.

4. おわりに

分散処理システムにおける高信頼化基礎技術についてまとめた. なにぶん, 多岐にわたる技術分野であるので, そのすべてをカバーできなかったが, 代表的な, また特に最近話題となっている技術は包含したつもりである.

ネットワーク及びそれを利用した各種システムが急速に社会のインフラストラクチャ化している現在, システムの障害は社会への計りしれない損失を与える可能性を秘めている. 最近では国際標準化の土俵でも, CCR (Commitment, Concurrency and Recovery)[13] に代表されるように活発に検討されるようになってきた. このような状況のもと, さらなる高信頼化基礎技術の充実と応用への展開を期待するものである.

参考文献

1) Randell, B., Lee, P. A. and Treleaven, P. C.: Reliability Issues in Computing Systems Design, Comput. Surv., Vol. 10, No. 2, pp. 123-165 (Jan. 1978).
2) Kohler, W. H.: A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems, Comput. Surv., Vol. 13, No. 2, pp. 149-183 (June 1981).
3) Bernstein, P. A. and Goodman, H.: Concurrency Control in Distributed Database Systems, Comput. Surv., Vol. 13, No. 2, pp. 185-221 (June 1981).
4) 高平, 鈴木: 分散型データベース技術, 情報処理, Vol. 23, No. 10, pp. 931-938 (Oct. 1982).
5) Lampson, B. and Sturgis, H.: Crash Recovery in a Distributed Data Storage System, Xerox Palo Alto Research Center, Technical Report, pp. 1-15 (1976).
6) Lomet, D, B.: Process Structuring, Synchronization and Recovery Using Atomic Actions, SIGPLAN, Vol. 12, No. 3, pp. 128-137 (Mar. 1977).
7) Davies, C. T. Jr.: Data Processing Spheres of Controls, IBM Syst. J., Vol. 17, No. 2, pp. 179-198 (1978).
8) Bjork, L. A.: Recovery Semantics for a DB/DC System, ACM Annual Conf., pp. 142-146 (Oct. 1973).
9) Jalote, R. and Campbell, R.: Atomic Actions for Fault-Tolerance Using CSP, IEEE Trans. Softw. Eng., Vol. SE-12, No. 1, pp. 59-68 (Jan. 1986).
10) Taylor, D. J.: Concurrency and Forward Recovery in Atomic Actions, ibid., pp. 69-78 (Jan. 1986).
11) Reed, D. P.: Implementation Atomic Actions on Decentralized Data, ACM Trans. Comput. Syst., Vol. 1, No. 1, pp. 3-23 (Feb. 1983).
12) Liskov, B. and Scheifler, R.: Guardians and Actions: Linguistic Support for Robust, Distributed Programs, ACM Trans. Program Lang. Syst., Vol. 5, No. 3, pp. 381-404 (July 1983).
13) 若山: 上位層のサービスとプロトコル・応用層, 情報処理, Vol. 26, No. 4, pp. 368-372 (Apr. 1985).
14) Lindsay, B. G. et al.: Notes on Distributed Databases, IBM Research Report, RJ 2571 (33471) (1971).
15) Colliat, G. and Bachman, C.: Commitment in a Distributed Databases, IFIP Working Conf., pp. 107-121 (June 1979).
16) Garcia-Molina, H.: Elections in a Distributed Computing System, IEEE Trans. Comput., Vol. C-31, No. 1, pp. 48-59 (Jan. 1982).
17) Garcia-Molina, H. and Barbara, D.: How to Assign Votes in a Distributed System, J. ACM, Vol. 32, No. 4, pp. 841-860 (Oct. 1985).
18) Lamport, L., Shostak, R. and Pease, M.: The Byzantine Generals Problem, ACM Trans. Program Lang. Syst., Vol. 4, No. 3, pp. 382-401 (July 1982).
19) Pease, R., Shostak, R. and Lamport, L.: Reaching Agreement in the Presence of Faults, J. ACM, Vol. 27, No. 2, pp. 228-234 (Apr. 1980).
20) Strong, H. R. and Dolev, D.: Byzantine Agreement, Proc. Compcon Spring, pp. 77-81 (1983).
21) Rabin, M. O.: Randomized Byzantine Generals, Proc. 24 th Annu. Symp. Foundations of Comput. Sci., pp. 403-409 (Nov. 1983).
22) Chang, J. M. and Maxemchuk, N. F.: Reliable Broadcast Protocols, ACM Trans. Comput. Syst., Vol. 2, No. 3, pp. 251-273 (Aug. 1984).
23) Lamport, L.: Using Time instead of Timeout for Fault Tolerant Distributed Systems, ACM Trans. Program Lang. Syst., Vol. 6, No. 2, pp. 254-280 (Apr. 1984).
24) Cristian, F., Aghili, H. and Strong, R.: Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement, Proc. 15 th Annu. Symp. on Fault-Tolerant Computing, pp. 200-206 (1985).
25) Hasegawa, S. and Liu, J. W. S.: A Reliable Token-Driven Process Synchronization Algo-

rithm, Proc. 6th Int. Conf. on Dist. Computing Syst., pp. 598-604 (1986).

26) Tanenbaum, A. S.: Computer Networks, Prentice-Hall, Inc. (1981).

27) Dalal, Y. K. and Metcalfe, M.: Reverse Path Forwarding of Broadcast Packets, Comm. ACM, Vol. 21, No. 12, pp. 1040-1048 (Dec. 1978).

28) Segall, A. and Awerbuch, B.: Reliable Broadcast Protocol, IEEE Trans. Commun., Vol. COM-31, No. 7, pp. 896-901 (July 1983).

29) Tajibnapis, W. D.: A Correctness Proof of a Topology Information Maintenance Protocol for Distributed Computer Networks, Comm. ACM, Vol. 20, No. 7, pp. 477-485 (July 1977).

(昭和61年11月27日受付)