データベース・システム 78−12
（1990. 7. 20）

分散データベースシステムの制御方式に関する一考察

根岸 和義　新田 淳　木保 清隆　神田 基博　木村 淳美
（株）日立製作所 システム開発研究所　同左 ソフトウェア工場

分散データベースシステムにおいてそのプロトコルの正当性の検証は重要な課題である。本報告では、有限オートマトン（ＦＳＡ）により記述された複数のプロトコルマシンで定義されているプロトコルの障害発生時を含む検証方式に関して述べる。各ＦＳＡに障害発生時の遷移を明示的に追加し、その組合せのＦＳＡを分析して、不当状態（コミット、ロールバックの混在した状態）や行き止まり状態（所期の最終状態に遷移できない状態）が発生しないことを検証した。ＦＳＡの組合せに際して状態の直積でなく、異なる状態の組合せのみを考えることにより、サーバマシンの台数とは独立に上記の検証を可能とした。

A Study of Control Methods for Distributed Database Systems

Kazuyoshi Negishi* Jun Nitta* Kiyotaka Kiyasu* Motohiro Kanda*
Atsumi Kimura**
*Systems Development Lab., Hitach Ltd. **Software Works, Hitachi Ltd.
1099 Ohzenji Asao-ku Kawasaki-si 215, JAPAN

Verification of communication protocol is important for distributed database systems. This paper discribes a verification method of the protocol. That protocol which is defined by each finite state automata(FSA) is extended to include state transitions to failure states. We examine these FSA to exclude invalid states which include both commit state and rollback state, and to exclude halt states which has no transition path to final states. Reduced FSA is used to analize the protocol independently of the number of database servers.

１．まえがき

近年、大型システムの発達とともに、計算機システムの処理能力に対するニーズも増大している。これに対処する手段の一つとして、複数の計算機システムを通信回線により連結し、互いのデータベースを相互にアクセス可能とした分散データベースシステムに対するニーズも高まっている。また、異機種の計算機システムにより分散データベースシステムを構成する際に、その相互接続を保証するプロトコルの標準化がＯＳＩとして推進されている。

従来、このプロトコルの定式化には、主に有限状態オートマトン（ＦＳＡ）によるプロトコルマシンとその直積によるＦＳＡが用いられてきた[(1)]。ＦＳＡによる定式化は主に正常動作を記述し、通信障害に対しては、通信を最初からやり直す事を想定したものが多かった。また、ＯＳＩのＴＰ[(2)]では、ジャーナル取得ポイントの記述はあるが回復方法は明示していない。

従来の方式の問題点は、ノード障害の回復処理がプロトコルの対象外のローカルマターとして規定から外されているため、ノード障害の回復処理において不正状態（各ノードのコミット、ロールバックが混在した終了状態）を生じないことの検証が困難である点である。従って、異機種接続時のノード障害からの回復時に矛盾を生じないことの保証ができない。また、複数ノードを相手とするプロトコルマシンが行き止まり状態（終了状態への遷移ができない　状態）を生じないことの検証をしようとしても、何個のノードを想定した検証を行えば十分であるかが不明であるという問題もあった。

そこで我々は、ノードにおけるジャーナル取得のタイミング、ノード障害の回復処理を含めたプロトコルマシンを想定し、ＦＳＡによりモデル化することにした。また、プロトコルマシンの状態の直積でなく、状態の種別を考え、状態の組合せによるＦＳＡを解析する方策を考案した。

我々の対象とするシステムは、クライアント、サーバモデルにより表現される分散データベースシステムである。クライアントのマシンは、全体を制御するメインマシンとサーバ対応の個別マシンから成る。各ノードは、独立にジャーナルを取得し、ノード障害に備えるシステムである。このシステムにおいて、ノードの障害発生、および回復をＦＳＡの状態遷移により、モデル化し、不当状態への遷移を組合せマシンによりチェックする方策を取った。プロトコルマシンのうちサーバマシン対応に設定される部分をまず直積マシンとして統合し、このマシンとシステム全体を統括するプロトコルマシンとのメッセージのやりとりを組合せマシンとして、分析する。その際に、サーバノード対応のプロトコルマシンを異なる状態の組合せにより表現し、モデルを簡単化するとともに、プロトコルの検証に必要なノード数の下限値をもとめる事を可能とした。

本報告のシステムの特徴は、ノード障害を含んだＦＳＡによるプロトコルマシンのモデル化と、状態組合せマシンによるシステムのモデル化である。

２．モデルの概要

我々はＯＳＩのＴＰ処理を想定して検討を行っているが、ここでは評価方式を明確にするため、サーバ、クライアント間のトランザクション処理に着目し、下記のように簡略化したモデルを検討した。本報告で想定するプロトコルマシンの構成を図１に示す。システムはクライアントノード及びｎ台のサーバノードより構成される。クライアントノードには、全体を統括するメインマシンと、サーバノードに対するインタフェイスとなる個別マシンがある。一方サーバノー

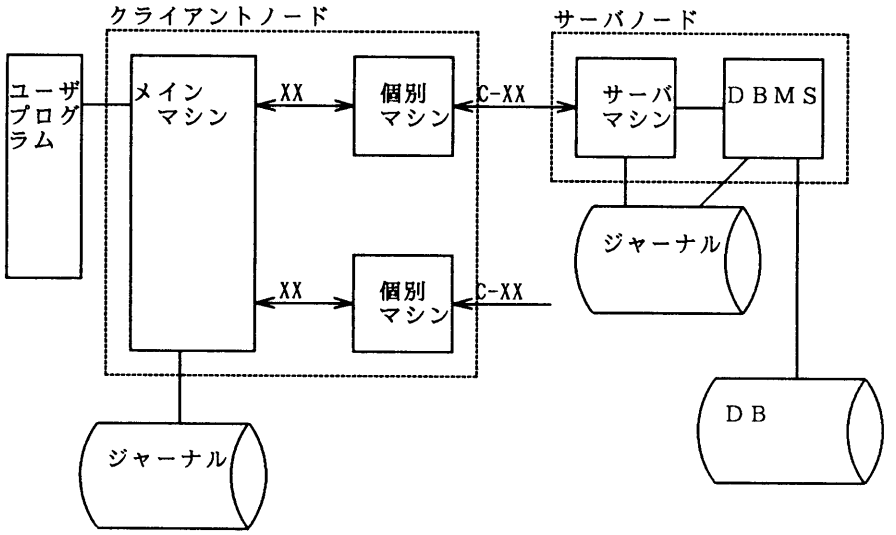


図１　プロトコルマシンの関連図

ドにはサーバマシンがある。各プロトコルマシンはメッセージにより互いに連絡する。

２．１　ジャーナルの取得

また、メインマシンおよびサーバマシンは障害時の回復のためにジャーナルレコードをディスク装置に出力する。個別マシンはジャーナル出力回数削減のため、個別にジャーナルを取得することはせず、障害発生時の回復はメインマシンのジャーナルを利用して行う。

ジャーナルは各プロトコルマシンの特定の状態で取得される。本モデルではプロトコルマシンの状態遷移中の障害発生はないものとし、特定の状態にある時に障害に対応する状態遷移を発生させることで障害発生をモデル化している。障害発生のタイミングと、ジャーナル取得のタイミングは独立であり、その前後関係により、同一状態から異なった状態へ遷移することによりこれを表現してる。

２．２　メッセージの伝達モデル

メインマシンと個別マシン、個別マシンとサーバマシン間のメッセージは各プロトコルマシン間で同時には一つのみ存在する（半二重通信方式を想定）。また、メッセージの消失、追越は想定しない。ノード障害の発生時、受信されていないメッセージは削除される。

個別マシンとサーバマシン間では、ノードダウン発生時、相手ノードに異常終了（Ｃ－ＡＢＴ）メッセージが送られる。ノードが障害より回復し、ノード間のコネクションが再開した時、コネクション再開（Ｃ－ＣＯＮ）メッセージがクライアントノードへ送信される。ただし、プロトコルマシンは同期送受信を前提としており、上記Ｃ－ＡＢＴメッセージおよびＣ－ＣＯＮメッセージは、相手のプロトコルマシンが受信状態となった時に受信される。また、ノードに障害が発生した場合の回復は時間を要し、Ｃ－ＡＢＴメッセージが受信される以前にＣ－ＣＯＮメッセージが発生することはないものとする。

メインマシンと個別マシンの間では、メインから個別には常に同一の内容を全個別マシンへ送信する。これは、トランザクションの処理開始時に関連する全ノードへトランザクション開始を連絡する方式を想定している。また、トランザクションに関連しないノードに関してはモデル化の対象外とする。また、個別マシンから、メインマシンへのメッセージは、下記のいずれかとする。

（１）全個別マシンから同一のメッセージを受信する。

（２）全個別マシンから二種類のメッセージのいずれかを受信する。ただし、各メッセージを少なくとも一つの個別マシンから受信するものとする。

２．３　プロトコルマシンの詳細

我々の想定するプロトコルマシンの状態遷移図を図２，図３，図４に示す。

（１）メインプロトコルマシン

図１はメインマシンの状態遷移図である。「ＴＲなし」状態から処理を開始し、トランザクション開始メッセージ（ＢＧＮ）を個別マシンへ送信し、「トランザクション開始処理中（ＴＲ開始処理中）」状態になり、ＢＧＮ応答メッセージを受信すると、「トランザクション実行中（ＴＲ実行中）」となる。個別マシンにコミット準備（ＰＰＲ）メッセージを送信すると、「コミット準備処理中（ＰＰＲ処理中）」状態となり、レディ（ＲＤＹ）メッセージを受信すると、

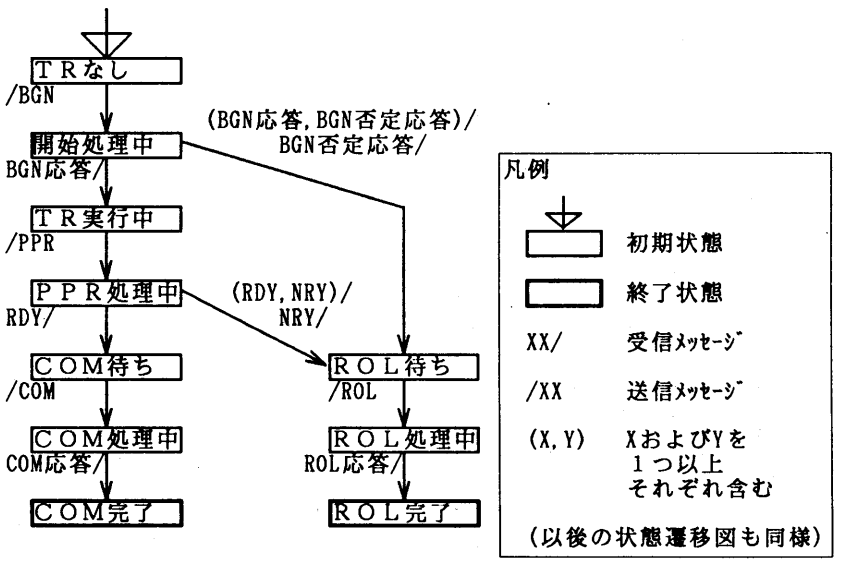


図２　メインマシンの状態遷移図

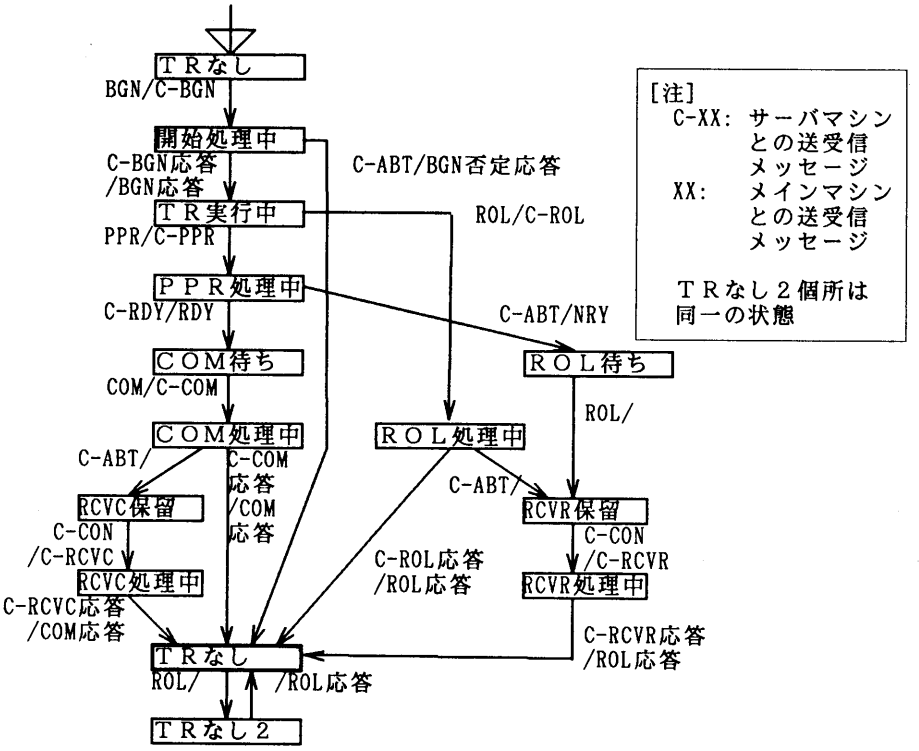


図３　個別マシンの状態遷移図

「コミット待ち（ＣＯＭ待ち）」状態となる。個別マシンにコミット（ＣＯＭ）メッセージを送信すると、「コミット処理中（ＣＯＭ処理中）」状態となり、ＣＯＭ応答メッセージを受信すると「コミット完了（ＣＯＭ完了）」状態となる。ＢＧＮメッセージにたいして、一つ以上の個別マシンからＢＧＮ否定応答が返されると、「ロールバック待ち（ＲＯＬ待ち）」状態となり、ロールバック（ＲＯＬ）メッセージを送信して、「ロールバック処理中（ＲＯＬ処理中）」状態となる。そして、ロールバック応答（ＲＯＬ応答）メッセージが返されると、「ロールバック完了（ＲＯＬ完了）」状態となる。また、ＰＰＲメッセージに対して、一つ以上の個別マシンからノットレディ（ＮＲＹ）メッセージが返されると、ＲＯＬメッセージを送信して上記と同様に「ＲＯＬ処理中」状態となる。ＣＯＭメッセージ、ＲＯＬメッセージに対してのノード障害は個別ノードにより吸収する方針のため、メインマシンではＣＯＭ否定応答およびＲＯＬ否定応答は想定しない。メインマシンの開始状態は「ＴＲなし」状態、終了状態は、「ＣＯＭ完了」状態および「ＲＯＬ完了」状態である。本報告では、クライアントノードの障害は対象外としているため、メインマシンのジャーナルに関するこれ以上の説明は省略する。

（２）個別プロトコルマシン

図３は個別マシンの状態遷移図である。個別マシンは、ほぼメインマシンど同様の状態遷移をおこなう。ただし、サーバノードのダウンのためＣ－ＡＢＴメッセージを受信した場合、下記のように特別の処理を行う。

（ａ）「開始処理中」状態で、Ｃ－ＡＢＴメッセージを受信した場合

メインマシンへＢＧＮ否定応答を返すとともに、「ＴＲなし」状態へ遷移する。以後、メインマシンからのＲＯＬメッセージに対して、ＲＯＬ応答メッセージを返送することで処理を終える。

（ｂ）「ＰＰＲ処理中」状態において、Ｃ－ＡＢＴメッセージを受信した場合

メインマシンにＮＲＹメッセージを返送するとともに、「ロールバック待ち（ＲＯＬ待ち）」状態に遷移する。以後、メインマシンからのＲＯＬメッセージに対して、「リカバーロールバック保留（ＲＣＶＲ保留）」状態に遷移してサーバノードの回復を待つ。サーバノードの回復時、Ｃ－コネクション再開メッセージを受信すると、リーカバーロールバック（Ｃ－ＲＣＶＲ）メッセージをサーバに送り、「リカバーロールバック処理中（ＲＣＶＲ処理中）」状態に遷移する。サーバからＣ－ＲＣＶＲ応答メッセージを受信すると、メインマシンにＲＯＬ応答メッセージを返送するとともに、「ＴＲなし」状態に遷移

して処理を終了する。
（ｃ）「ＣＯＭ処理中」状態でＣ－ＡＢＴメッセージを受信した場合
個別マシンは「リカバーコミット保留（ＲＣＶＣ保留）」状態に遷移してサーバノードの障害回復を待つ。サーバノードからのＣ－ＣＯＮメッセージによりリカバーコミット（Ｃ－ＲＣＶＣ）メッセージをサーバマシンへ送信し、「リカバーコミット処理中（ＲＣＶＣ処理中）」状態へ遷移する。サーバノードからＣ－ＲＣＶＣ応答を受信すると、メインマシンにＣＯＭ応答を返送するとともに、「ＴＲなし」状態に遷移して処理を終了する。
（ｄ）「ＲＯＬ処理中」状態でＣ－ＡＢＴメッセージを受信した場合
個別マシンは「ＲＣＶＲ保留」状態に遷移し、先程と、同様にサーバマシンからのＣ－ＣＯＮメッセージを待つ。個別マシンの開始状態、および終了状態はいずれも「ＴＲなし」状態である。
（３）サーバプロトコルマシン
図４はサーバプロトコルマシンの状態遷移図である。サーバマシンもほぼメインマシンと同様の状態遷移を行う。本マシンでは、「ＰＰＲ処理中」状態、「ＣＯＭ処理中」状態、「ＲＯＬ処理中」状態、「ＣＯＭ完了」状態、「ＲＯＬ完了」状態、「ＲＣＶＣ処理中」状態、「ＲＣＶＲ処理中」状態において、ジャーナルを取得する。これに従って、障害発生時の「ダウンＴＲなし」状態、「ダウン未決凍結」状態、「ダウンＣＯＭ完了」状態、「ダウンＲＯＬ完了」状態への状態遷移が図４のように定義される。先に述べたように障害発生時に同一の状態から遷移先が複数存在するケースもある。
障害回復時、「ダウン未決凍結」状態からの回復時は「未決凍結」状態へ遷移するとともに、Ｃ－ＣＯＮメッセージがクライアントノードの個別マシンへ送られる。「ダウンＴＲなし」、「ダウンＣＯＭ完了」、「ダウンＲＯＬ完了」状態は、それぞれ、「ＴＲなし」、「ＣＯＭ完了」、「ＲＯＬ完了」状態に遷移するとともに、同様にＣ－ＣＯＮメッセージがクライアントノードの個別マシンへ送信される。「未決凍結」状態に対して、Ｃ－ＲＣＶＣメッセージ、あるいはＣ－ＲＣＶＲメッセージを受信した場合、それぞれ、ＲＣＶＣ処理中、およびＲＣＶＲ処理中を経由して、Ｃ－ＲＣＶＣ応答メッセージ、Ｃ－ＲＣＶＲ応答メッセージを送信するとともに、「ＣＯＭ完了」状態、あるいは、「ＲＯＬ完了」状態へ遷移する。サーバマシンの開始状態は、「ＴＲなし」状態、終了状態は「ＴＲなし」状態、「ＣＯＭ完了」状態、「ＲＯＬ完了」

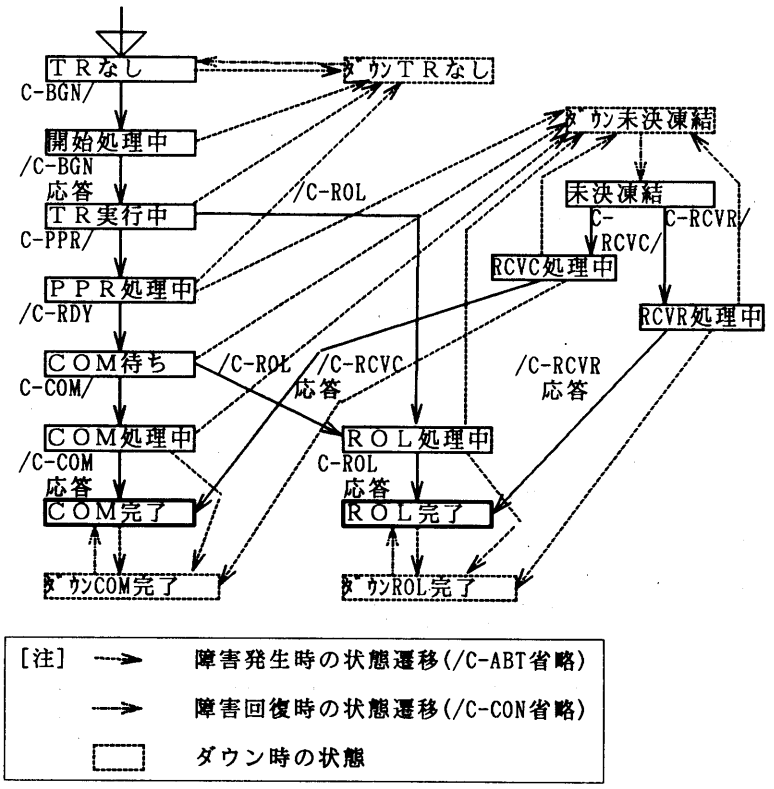


図４　サーバマシンの状態遷移図

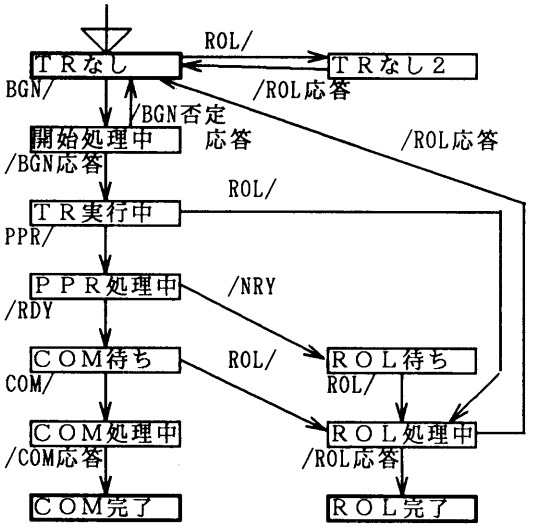


図５　サーバ直積マシンの状態遷移図

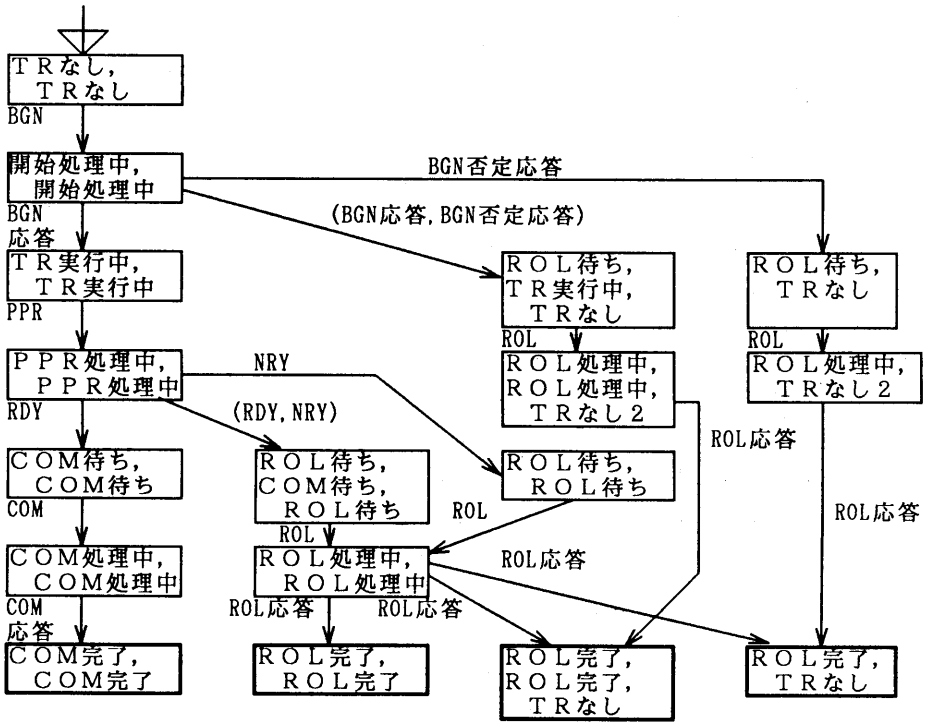


図６　組合せマシンの状態遷移図

状態である。

3．障害回復に関するプロトコルの正当性検証

プロトコルマシンの検証には、各マシンの状態の直積とメッセージを組合せた新しいＦＳＡ（サーバ直積マシンと呼ぶ）を考えその状態遷移を考える。まず、ｎ台のサーバノードに対してｎ個の個別マシンとサーバマシンを統合したプロトコルマシンを考え、そのマシンとメインマシンとのメッセージのやりとりに着目して状態の統合を行う。図５に、図３と図４のマシンを合成し状態の統合を行ったサーバ直積マシンの状態遷移図を示す。この図において、開始状態は「ＴＲなし」状態であり、終了状態は「ＴＲなし」状態、「ＣＯＭ完了」状態、「ＲＯＬ完了」状態のいずれかである。

次に、メインマシンと上記サーバ直積マシンとの直積マシンを考える。この時、サーバ直積マシン側は、個別のマシンの状態ではなく、状態の種別の組合せを考える。図６の例でこれを説明する。まず、初期状態は、メインマシン、サーバ直積マシンとも「ＴＲなし」状態であるから、組合せマシンの初期状態は（「ＴＲなし」，（「ＴＲなし」））である。ここで、（Ｓ０，Ｓ１，．．，Ｓｍ）は、Ｓ０がメインマシンの状態、Ｓ１，．．Ｓｍがｍ通りのサーバ直積マシンの状態の組合せを表す。メインマシンからサーバ直積マシンへＢＧＮメッセージを同時に送信すると組合せマシンの状態は、（「開始処理中」，「開始処理中」）となる。次に、サーバマシンがダウンすることなく開始処理を終了すると全てのサーバ直積マシンからＢＧＮ応答が返り、組合せマシンは状態（「ＴＲ実行中」，「ＴＲ実行中」）に遷移する。しかし、いくつかのサーバノードでダウンが発生した場合、当該サーバ直積マシンはＢＧＮ否定応答を返すから、組合せマシンの状態は、（「ＲＯＬ待ち」，「ＴＲ実行中」，「ＴＲなし」）に遷移する。また、すべてのノードでダウンが発生した場合、すべてのノードがＢＧＮ否定応答を返すので、組合せマシンの状態は、（「ＲＯＬ待ち」，「ＴＲなし」）に遷移する。以下同様にして、図６が得られる。ここで、組合せマシンの終了状態は、

（「ＣＯＭ完了」，「ＣＯＭ完了」）
（「ＲＯＬ完了」，「ＲＯＬ完了」）
（「ＲＯＬ完了」，「ＲＯＬ完了」，
「ＴＲなし」）
（「ＲＯＬ完了」，「ＴＲなし」）

のいずれかである。従って、「ＴＲなし」と「ＲＯＬ完了」を同一視すれば、この組合せマシンにおいて、プロトコルの規約に矛盾する不正状態への遷移は発生しない。また、系１．１より対応する直積マシンにおいても不正状態への遷移は発生しない。

次に、組合せマシンと直積マシンとの関係を調べる。定理１より、直積マシンの状態遷移に対応する組合せマシンの状態遷移が存在することは明らかである。組合せマシンの状態遷移に対応する直積マシンの状態遷移が存在するかを分析するためには、組合せマシンの各状態に対して直積マシン上で当該状態に対応する状態に遷移するために最小限何台のサーバマシンが必要かをアルゴリズム１により分析することが必要である。分析の結果を図７に示す。図７ではステップ１のＤＮｉを（）により表し、ステッ

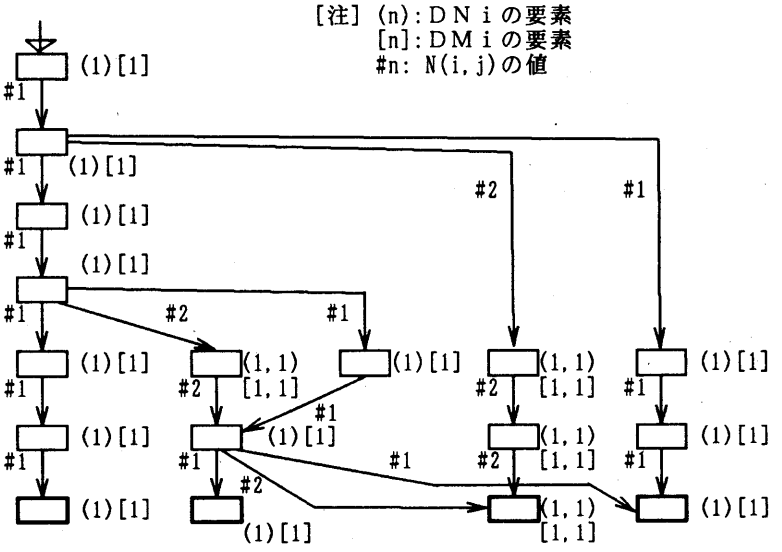


図７　アルゴリズム１，２の評価結果

プ2のDMiを［］により表す。ステップ3により、この例では、行き止まり状態がないことがわかる。図7にアルゴリズム2を適用すると#nの最大値が2であることから本プロトコルマシンでは最小限2台のサーバマシンがあれば全ての組合せを検証できることがわかる。ここで、DMiを定義する過程で、ノードの障害発生時にしか遷移しないものが含まれている可能性がある。これを避けるにはステップ2において、直積マシンの遷移として正常時の遷移のみを対象とするようアルゴリズムを変更すればよい。ここでは、簡単のためその可能性は考慮しない。

4．まとめ

本報告では、分散データベースシステムのプロトコルマシンモデルに対して、障害回復処理の定義を含め、障害回復処理の正当性を示した。また、プロトコルの検証をノード別の状態ではなく、状態の種別に対して行うことにより、ノード数と独立にプロトコルが行き止まり状態を生じないことの検証を行うアルゴリズムを示した。さらに、プロトコルの検証に必要なノード数を求めるアルゴリズムを示した。

本方式により、分散データベースシステムのように1対nの関係にあるプロトコルシステムのnと独立したプロトコル検証が可能となった。また、プロトコル実証時の必要なノード数を求めることができた。

参考文献

[1] Dale Skeen,"Formal Model of Crash Recovery in a Distributed System",IEEE Trans. on Software Eng. Vol.SE-9,No.3 May 1983 pp219-228.

[2] ISO/IEC DIS 10026 Information Processing Systems-Open Systems Interconnection-Distributed Transaction Processing

[3] J.E.ﾎｯﾌﾟｸﾛﾌﾄ,J.D.ｳﾙﾏﾝ,"言語理論とオートマトン",ｻｲｴﾝｽ社(1971)

付録　記号の定義とアルゴリズム、定理

文献［3］の記法に基づき、n台のサーバ直積マシンを下記で表す。

$M=(X, Q, \delta, q_0, F)$

ここで、

X：マシンの入力記号の集合
Q：マシンの状態の集合
δ：マシンの状態遷移関数の集合
q：マシンの初期状態
F：マシンの終了状態の集合

とする。

次に、X，Q，δにアルファベット順を仮定し、AをX，Q，δのいずれかに固定した時、

$(A)^n$：Aの要素の長さnの列

とする。

長さnの列$a_1..a_n \in (A)^n$に対して、

$U(a_1..a_n)=$（$a_1..a_n$のうち重複するものを除きアルファベット順に並べ換えたもの）

なる関数を考える。次に、

$(A)^+$：Aの要素の列の集合

とした時、Uを拡張して、

$U((A)^+)=\{U(a_1..a_n) \mid a_1..a_n \in (A)^+\}$

とする。これを用いて、Mと同様にメインマシンを下記で表す。

$M^0=(X^0, Q^0, \delta^0, q_0^0, F^0)$

ここで、

$X^0=U((X)^+)$

である。

M^0とMはその入出力メッセージをすべて入力記号と見なすことにより、ＦＳＡを定義している。以下の直積マシン、および組合せマシンでは、M^0とn個のMが各々$U(x^1..x^n)$，および$x^1..x^n$を入力した時、メッセージの送受信が成立したと見なす。

サーバ直積マシンとメインマシンの直積マシンを下記のように定義する。

$M^p=(X^p, Q^p, \delta^p, q_0^p, F^p)$

ここで、

$X^p=(X)^n$
$Q^p=Q^0\times(Q)^n$
$\delta^p: Q^p\times X^p\rightarrow Q^p$
$q_0^p=q_0^0q_0..q_0$
$F^p=F^0\times(F)^n$

である。ただし、

$\delta^p(q_i^0q_i^1..q_i^n, x^1..x^n)=(q_j^0q_j^1..q_j^n)$

となるのは、

$\delta^0(q_i^0, U(x^1..x^n))=q_j^0$　かつ、
$\delta(q_i^k, x^k)=q_j^k$　$(k=1\sim n)$

の時である。

また、サーバ直積マシンとメインマシンの組合せマシンを下記のように定義する。

$M^s=(X^s, Q^s, \delta^s, q_0^s, F^s)$

ここで、

$X^s=U((X)^+)$
$Q^s=Q^0\times U((Q)^+)$
$\delta^s: Q^sX^s\rightarrow Q^s$
$q^s=(q_0^0q_0)$
$F^s=F^0\times U((F)^+)$

である。ただし、

$\delta^s(U(q_i^0q_i^1..q_i^n), U(x^1..x^n))=U(q_j^0q_j^1..q_j^n)$ となるのは、
$\delta^p(q_i^0q_i^1..q_i^n, x^1..x^n)=q_j^0q_j^1..q_j^n$

の時に限る。

上記の直積マシンと、組合せマシンに対して、以下の定理が成り立つことを証明した。

［定理1］

直積マシンにおいて、初期状態から状態$q_i^0q_i^1..q_i^n$への状態遷移が存在する時、組合せマシンにおいて初期状態から対応する状態$U(q_i^0q_i^1..q_i^n)$への状態遷移が存在する。

［証明］

定義より、直積マシンの状態及び状態遷移に対応して、組合せマシンの状態および状態遷移が必ず存在する。従って、直積マシンにおいて、初期状態からある状態への遷移が存在するなら、それに対応して直積マシンにおいて、初期状態から対応する状態への遷移が存在する。［証明終］

次に、不正状態と行き止まり状態を下記のように定義する。

不正状態：個別のサーバ直積マシンの終了状態の集合を複数のグループに分けた時、直積マシンの終了状態のうちこれら複数のグループの二つ以上のグループに属する終了状態を含むものを不正状態とする。

行き止まり状態：ＦＳＡにおいて、初期状態から遷移可能な状態で、その状態から終了状態への遷移ができない状態を行き止まり状態とする。

この時、以下の系が成立する。

［系１．１］

組合せマシンが初期状態から不正状態への遷移を含まない時、対応する直積マシンも初期状態から不正状態への遷移を含まない。

証明は定理１より明らかである。

サーバ直積マシンｎ個とメインマシンを組合せた場合、組合せマシンに行き止まり状態が存在しない場合、直積マシンにも行き止まり状態が存在しないかどうかをチェックする以下のアルゴリズムを考案した。

［アルゴリズム１］

［ステップ１］

組合せマシンの各状態に対して、下記のＤＮｉを求める。

ＤＮｉ：組合せマシンにおいて初期状態から状態ｉ＝${}_0q_{i1}q_i..{}_mq_i$への遷移に対応するｎが最小な直積マシンでの遷移を考えた時、直積マシンでの対応する状態$q_i^0q_i^1..q_i^n$に対して定義される

$dn_i=({}_1qn_i,...,{}_mqn_i)$の集合をＤＮｉとする。ここで、

${}_kqn_i$：直積マシンの上記状態における${}_kq_i$の個数、とする。

［ステップ２］

組合せマシンの各状態ｉに対して、下記のＤＭｉを求める。

ＤＭｉ：組合せマシンにおいて状態ｉ＝${}_0q_{i1}q_i..{}_mq_i$から最終状態への遷移に対応するｎが最小な直積マシンでの遷移を考えた時、直積マシンでの上記状態に対応する状態$q_i^0q_i^1..q_i^n$に対して、ステップ１と同様にdm_iを定義しその集合をＤＭｉとする。

［ステップ３］

ＤＮｉ，ＤＭｉに対して、各々必要な最小サーバ直積マシン数をＮ(ＤＮｉ)，Ｎ(ＤＭｉ)とする。サーバ直積マシン数ｎに対して、Ｎ(ＤＮｉ)≧ｎ＞Ｎ(ＤＭｉ)の時、状態ｉに対応する直積マシンの状態は、行き止まり状態である。また、ｎ≧Ｎ(ＤＮｉ)≧Ｎ(ＤＭｉ)であっても、$dn_i\in$ＤＮｉに対して

$\neg\exists dm_i\in$ＤＭｉ（$dn_i\supseteq dm_i$）

であれば、上記と同様に行き止まり状態である。ただし、

$dn_i\supseteq dm_i$：for all k（${}_kqn_i\geqq{}_kqm_i$）

である。

次に、組合せマシンの全遷移に対応した直積マシンの遷移が存在するためにはｎがいくつ以上であれば良いかを求めるアルゴリズムを考案した。

［アルゴリズム２］

組合せマシンの隣合った状態ｉ，ｊに対して、対応する直積マシンの状態遷移で下記を満たす最小のｎ(ｉ，ｊ)を求める。

$q_i^p\in Q^p$，$q_j^p\in Q^p$，

q_i^p，q_j^pに対してdm_i，dn_jを定義した時、

$\exists dm_i'\in$ＤＭｉ（$dm_i'\subset dm_i$），

$\exists dn_j'\in$ＤＮｊ（$dn_j'\subset dn_j$）

を満たす。

組合せマシンの全ての隣り合った状態ｉ，ｊについて、ｎ(ｉ，ｊ)を求め、その最大値をｍａｘｎとする。ｍａｘｎは組合せマシンの全ての遷移を実現するのに必要な最小限のサーバ直積マシン数である。