システムソフトウェアと
オペレーティング・システム 74−21
マルチメディア通信と分散処理 81−21
（1997. 2. 28）

システムの構成変更に対するやわらかいクラスタ間通信プロトコル

安田 昌史　島 健司　桧垣 博章　滝沢 誠

東京電機大学理工学部経営工学科

現在の情報システムは、通信網により複数の計算機を相互接続した分散型となってきている。このような分散システムでは、利用者の要求の変化に対応して分散アプリケーションをバージョンアップ(改版)することが求められる。従来、分散アプリケーションの改版はシステムを一時停止することによって実現されてきた。一時停止中は利用者がサービスを受けることはできない。特に、大規模システムでは規模の拡大にともない、改版する時間が増加し、システムの可用性が低下する。したがって、動作中の大規模システムを停止することなく改版するための技術が求められている。これに対して、動的改版手法およびその実現プロトコルが提案されている。しかし、このプロトコルでは新と旧のプロセスが一対一に対応する場合にのみ適用できる。全ての機能一つのプロセスで実現する集中型システムを同じ機能を、複数のプロセスで実現する分散型システムに改版できない。本論文では、新旧版プロセスの対応関係が一対多または多対一であっても適用できるように拡張した動的改版プロトコルを提案する。これにより、あるプロセスが持つ機能を複数のプロセスに分割したり、複数のプロセスが持つ機能を一つのプロセスに統合するといった、システムの構成変更をともなう改版が可能になる。

Flexible Inter-Cluster Communication Protocol to Absorb Configration Changes

Masashi Yasuda, Kenji Shima, Hiroaki Higaki, and Makoto Takizawa

Tokyo Denki University
E-mail {masa,sima,hig,taki}@takilab.k.dendai.ac.jp

Recently, the development of computer and communications technology has led to the development of large-scale distributed systems. These systems are frequently required to be upgraded in order to absorb the change of user's requirements. In the conventional upgrading methods, multiple processes are required to be suspended simultaneously. While the upgrading procedure is executed, the services are not provided for the users. Especially, it takes longer time to upgrade the large-scale distributed system. The inter-cluster communication protocol for the dynamic upgrading has been proposed so far. However, the protocol can be applied to the limited cases where each old process is always upgraded to one new process. In this paper, we propose a novel upgrading protocol which can be applied to the cases where one old process can be upgraded to multiple new processes or multiple old processes are upgraded to one new process. Thus, the upgrading method can be applied to more general cases than our previous protocol.

1 はじめに

近年、計算機技術や通信技術の発達によって、大規模な分散システムが構築されてきている。一般に大規模システムは長期間運用される。利用者要求とシステム環境の変化に追従できる機能を備えることが分散システムに求められる [12]。その一つとして、アプリケーションプログラムをバージョンアップ(改版)する機能がある [6]。ここでメールサービスなどのように、常時提供され続けるべきサービスのサーバプログラムを改版する場合を考える。このとき、改版の手続きによって、プロトコルエラーなどの不整合が発生してはならない。改版の手続きによって発生するプロトコルエラーには、未定義受信と通信デッドロックがある。未定義受信とは、アプリケーションプログラムが受理できないメッセージを受信することである。図1では、プロセス p は、q から $R = \{a, b, c\}$ に含まれるいずれかのメッセージを受信することになっているとする。しかし、q が $x \notin R$ を送信すると、p は x を受信できず、未定義受信となる。また、通信デッドロックとは、二つのプロセスが互いに相手から送信されるメッセージの受信待ちとなることである。図1では、p は q から x が送信されるのを待っており、q は p から y が送信されるのを待っている。このとき、p と q は互いに受信待ちとなり、通信デッドロックが発生している。

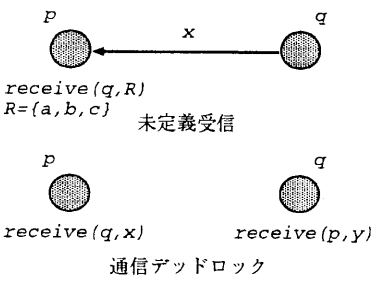

図 1: 未定義受信と通信デッドロック

一方、改版の手続きが実行されている最中でも、可能な限り利用者へサービスを提供し続けられることが望ましい。これまでに、システム全体を停止させることなく改版を行なう方法がいくつか提案されている [3, 7, 8, 10]。ここで、システムを構成するプロセス集合を P、改版対象のプロセスの集合を $P_u \subseteq P$ とする。CONIC システムで用いられている方法 [7, 8] では、プロセス集合 $P_s (P_u \subseteq P_s \subseteq P)$ に含まれる全てのプロセスを停止させてから P_u 内のプロセスを新しいアプリケーションプログラムを実行するプロセスに置き換えることによって、プロトコルエラーを発生することのない改版を実現することができる。P_s は P 内のプロセス間の依存関係と P_u とから一意に定めることができる。この方法では、アプリケーションプログラムを実行するプロセスが常に同一版のプロセスとしか通信しないため、改版によってプロトコルエラーが発生することはない。しかし、P_s 内の全てのプロセスを停止するための同期オーバヘッドを必要とする。特に、大規模システムに適用した場合には、ユーザがサービスを受けることができない時間が長くなるという問題点がある。

これに対して、文献 [3–5] で提案されている動的改版手法は、改版されるプロセス(旧版プロセス)と新しく置き換えられるプロセス(新版プロセス)とが一時的にシステム内に混在することを許している。これにより、複数のプロセスを同時に停止させずに改版を行える。同期オーバヘッドなくし、サービスを停止せずに提供できる。複数版の混在によって、異なる版のプロセスが通信するために、システム内でプロトコルエラーが発生することがある。チェックポイント設定とロールバック回復の機構を用意することによって、アプリケーションプログラムの実行に矛盾が発生することを防いでいる。

また、CONIC の方法は、プロセス間の依存関係が主従関係のみに限定されているクライアント・サーバ型のアプリケーションの改版のみにしか適用できない [1]。これに対して、文献 [3–5] の方法は、複数プロセスを同時に停止する必要がないため、クライアント・サーバ型以外の一般の対等型分散アプリケーション [13] に適用できる。しかし、旧版プロセスと新版プロセスの対応関係に注目してみると、文献 [3–5] で提案されているプロトコルでは、一つの旧版プロセスを一つの新版プロセスで置き換える場合にしか適用できない。ここで、図2に示す電子会議システムを考える。議題 α を議論している参加者 A, B, C, D が、二つの異なる議題 β と γ を議論する二つのグループに分かれる場合を考える。議題 β と γ の両方に参加したい A は、議題 β に参加するプロセスと議題 γ に参加するプロセスの二つに分割される必要がある。逆に、議題 β と議題 γ を一つの議題 α にまとめて議論する場合には、議題 β と γ の両方に参加している A' は、統合された A という一つのプロセスとして議題 α に参加する必要がある。

そこで本論文では、文献 [3–5] で提案されている動的改版プロトコルを拡張し、改版対象の新旧プロセス間の対応関係が一対多(機能統合)あるいは多対一(機能分割)の場合にも適用可能な新しいプロトコルを提案する。なお、本論文は以下のように構成されている。第 2 章では、動的改版手法の概要について述べる。第 3 章では、機能統合や機能分割に適用可能なクラスタ間通信プロトコルを提案する。第 4 章では、第 3 章で提案するプロトコルのメッセージ数と所要時間による評価を行なう。

2 動的改版手法

近年、大規模ネットワークを活用した通信システム、分散制御システム、会議システム、分散データベースといった分散アプリケーションは、自律的に動作、通信する複数のプロセスが協調動作することによって実現されている。このような分散アプリケーションは対等型分散アプリケーションと呼ばれ、プロセス間の依存関係には次のような特徴がある。

- プロセスは同時に他の複数のプロセスと依存関係を持つ。
- 依存関係を持つプロセスは動的に決定される。
- プロセスは他のプロセスと対等な立場で非同期的に通信する(関係に固定した方向性がない)。

このためそれぞれのプロセスにおいて停止可能な安定状態の獲得は難しい。また、停止対象プロセス集合 P_s 内のプロセス全てを停止することは困難である。

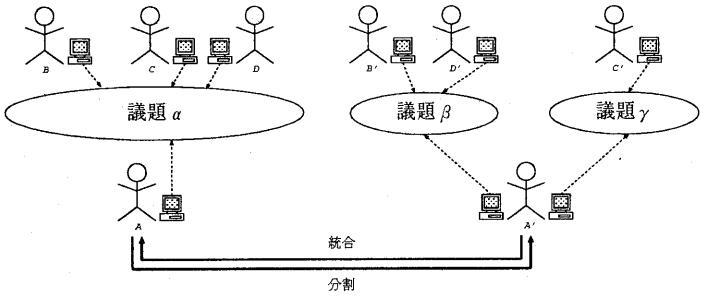

図 2: 電子会議システムにおける機能統合と機能分割

さらに、プロセス間の依存関係が複雑であるために P_s の決定が困難であり、決定可能であっても改版対象プロセス集合 P_u に対して、$|P_s| \gg |P_u|$ が一般に成り立つ。

これらの問題を解決するために動的改版手法が提案されている [3–5]。ここでは複数のプロセスを同時に停止することなく、非同期的な新版プロセスの起動を可能にしている。これによって、同期停止オーバヘッドの問題を解消するとともに対等型分散アプリケーションへの適用を可能としている。また、新旧版プロセスの混在によって発生し得るプロトコルエラー (未定義受信とデッドロック) に対しては、チェックポイント設定機構とロールバック回復機構を用意している。これによって、分散アプリケーション実行への影響は最小限になるように制御できる。以下に動的改版手法の概要を示す (図 3)。

1. システムは改版開始状態、過渡状態、改版終了状態、ロールバック状態の 4 つの状態をとる。改版開始状態では、旧版プロセスはそれぞれ一つのクラスタに属する。クラスタは、フォールトトレラントを実現するための複製されたプロセスの集合 [11] として用いられてきた。本論文では、改版を実現するための旧版プロセスとそれに対応する新版プロセスの集合としてクラスタを用いる。
2. 新版プロセス (p', q', r') は他と同期することなく対応するクラスタ内に起動される。このとき、旧版プロセス (p, q, r) は停止せず、バックアップとしてアプリケーションの実行を継続する。システムには、新旧版プロセスが並行動作している改版過程クラスタと旧版プロセスのみが動作している未改版クラスタが混在する過渡状態が形成される。
3. 改版過程クラスタでは、旧版プロセスが新版プロセスと同一の状態遷移が可能な範囲でアプリケーションの実行を継続する。ロールバック回復処理後のシステムが無矛盾な状態となるために、旧版プロセスは新版プロセスで起こったイベントと自プロセスで起こったイベントとを比較することによって、適切な状態にチェックポイントを設定する。
4. 過渡状態でプロトコルエラーを発生することなく、全ての新版プロセスが起動された場合には、旧版プロセスを停止して改版処理を終了する。
5. 過渡状態における通信でプロトコルエラーが発生した場合には、改版過程クラスタ内の新版プロセスを停止し、3 で設定されたチェックポイントから旧版プロセスがアプリケーションの実行を再開する。

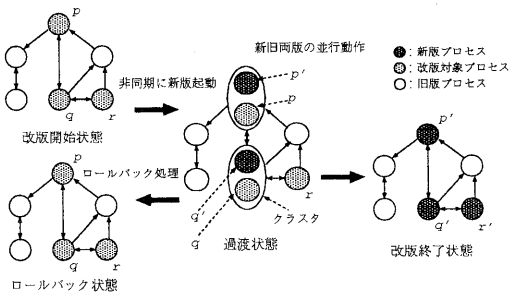

図 3: 動的改版の概要

3 提案手法

3.1 システムモデル

各プロセスは 3 階層で構成されているものとする [11]。上位層のアプリケーションレイヤでは、分散アプリケーションプログラムが実行される。送信イベントおよび受信イベント発生時には、クラスタ間通信レイヤの機能を用いてアプリケーションメッセージの交換を行なう。中間層のクラスタ間通信レイヤでは、提案するプロトコルが機能する。プロトコルエラーはこのレイヤで検出される。チェックポイント設定やロールバック回復の影響はこのレイヤで吸収されるため、上位のアプリケーションレイヤに対しては隠蔽される。下位層のプロセス通信レイヤは、プロセス間の一対一通信をサポートする。このレイヤの機能によって、上位のクラスタ間通信レイヤに対してはプロセス間に信頼性のある (信号の紛失、複製、書換えが起こらない)FIFO (First-in/First-out) 通信チャネルが提供されている。ここで以下を仮定する。

[仮定 1] 各プロセスにおけるアプリケーションプログラムの動作は決定的状態遷移機械 (DFSM) でモデル化できる。

[仮定 2] 同一版プロセス間の通信ではプロトコルエラーを発生しない。

[仮定 3] システムには 2 版のみが混在する。

[仮定 4] 新旧版プロセスの混在動作によって通信デッドロックは発生しない[1]。

[仮定 5] 全てのプロセスは、全ての新旧版プロセスの対応関係を知ることができる。

3.2 プロトコルが満たすべき性質

動的な改版を実現するプロトコルは、以下の性質を満たさなければならない。

[性質 1] 同一クラスタに属する新旧版プロセスのアプリケーションレイヤには、同一メッセージが同一順序で配付される。

[性質 2] 同一クラスタに属する新旧版プロセスからの送信メッセージが重複したり欠落したりすることはない。

[性質 3] 全ての未定義受信の発生を検出することができる。

[性質 4] 旧版プロセスが定めるチェックポイント群は、旧版プロセスのみからなるシステム動作として正当な広域状態を定める。つまり、ロールバック回復処理後の旧版プロセスの動作には、遅延を除いて過渡状態を経た影響が残らず、新旧版プロセスの並行動作をはじめから行なわなかった場合と同一の状態遷移を行なう。

3.3 チェックポイントプロトコル

過渡状態における旧版プロセスは、以下の条件のいずれかを満たすときチェックポイントを設定する。

1. 新版プロセスで発生したイベントと自プロセスで発生したイベントが異なる場合
2. チェックポイント設定済みのクラスタからアプリケーションメッセージを受信した場合

これによって、[性質 4] を満たすことができる [3-5]。本節では、これを一対多、多対一の改版で実現するプロトコルについて述べる。

3.3.1 一対多の動的改版

一対多 (機能統合) の改版におけるプロトコルを示す。ここでアプリケーションメッセージ M はクラスタ P からクラスタ Q へ送られるものとする。また、クラスタ P に属するの旧版プロセスの集合を、$G_P = \{p_{o1}, \ldots, p_{oN}\}$ とする。

送信イベントにおけるプロセスの動作 (図 4)

1. 新版プロセス p_n で送信イベントが発生したとき、以下の条件を満たすならば、3.4 で述べるロールバック回復手続きを行なう。

   Q が一対多の改版対象クラスタであり、新版プロセス q_n が未起動である。

2. p_n は Q の新版プロセス q_n に対して[2]、M を含むシステムメッセージ m_1 を送信する。また、G_P 内の全ての旧版プロセスに対して M を含むシステムメッセージ m_2 を送信する。
3. G_P 内の全ての旧版プロセスは、m_2 を受信した後、M が自プロセス内で発生した送信イベントで送るメッセージと同一であるか否かの情報を含むシステムメッセージ m_3 を p_n に送信する。
4. p_n は G_P 内の全ての旧版プロセスから m_3 を受信した後、その m_3 の内容が全て否であるならば、この送信イベントの直前にチェックポイントを設定することを指示するシステムメッセージ m_4 を G_P 内の全ての旧版プロセスに対して送信する。このとき、m_4 を受信した旧版プロセスは、この送信イベントの直前にチェックポイントを設定し、一時停止する。
5. m_3 の少なくとも一つに同一イベントであるという情報が含まれているならば、p_n はこの m_3 を送信した旧版プロセスのうちの一つを決定し (ここでは p_{o3} とする)、その旧版プロセス p_{o3} 対して、システムメッセージ m_6 の送信を促すシステムメッセージ m_5 を送信する。
6. m_5 を受信した p_{o3} は、m_6 を Q の旧版プロセス q_o に送信する。m_6 には G_P がチェックポイントを設定しているか否かの情報が含まれる。

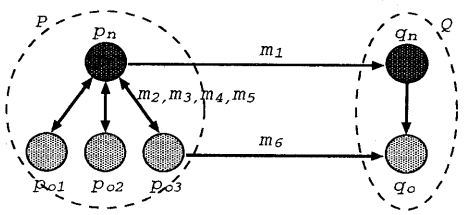

図 4: 一対多の改版における送信イベント

受信イベントにおけるプロセスの動作 (図 5)

1. アプリケーションメッセージ M を含むシステムメッセージ m_1 を受信した新版プロセス q_n は、これをメッセージキューに登録する。また、M を要求するシステムメッセージ m_3 を旧版プロセスから受信するまでの間、m_1 を保持する。
2. システムメッセージ m_2 を受信した旧版プロセス q_{oi} (ここでは q_{o1} とする) は新版プロセス q_n に対して、m_3 を送信する。
3. m_3 を受信した q_n は、q_{o1} に対して、M を含むシステムメッセージ m_4 を送信する。
4. m_4 を受信した q_{o1} は、この M が自プロセス内で発生した受信イベントで受理可能なメッセージであるならば M を受理する。そうでないならば、この受信イベントの直前にチェックポイントを設定し、一時停止する。
5. m_2 にチェックポイント設定済みの情報が含まれているならば、q_{o1} は対応する受信イベントの直前にチェックポイントを設定し、一時停止する。

3.3.2 多対一の動的改版

多対一 (機能分割) 改版におけるプロトコルを示す。

送信イベントにおける新旧版の動作 (図 6)

1. 新版プロセス p_{ni}(ここでは p_{n3} とする) で送信イベントが発生したとき、以下の条件を満たす

[1] 新旧版プロセス仕様間の差異が現れる最初のイベントは、ともに送信イベントあるいはともに受信イベントである [4]。

[2] ただし、q_n が未起動ならば q_o に対して送信する。

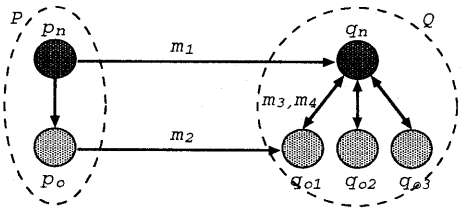

図 5: 一対多の改版における受信イベント

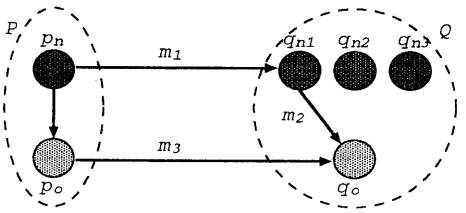

図 7: 多対一の改版における受信イベント

ならば、3.4 で述べるロールバック回復手続きを行なう。

Q が一対多の改版対象クラスタであり、新版プロセス q_n が未起動である。

2. p_{n3} は Q の新版プロセス q_n に対して[3]、M を含むシステムメッセージ m_1 を送信する。また、P 内の旧版プロセス p_o に対してアプリケーションメッセージ M を含むシステムメッセージ m_2 を送信する。
3. p_o は、m_2 の受信後、この M が自プロセス内で発生した送信イベントで送るメッセージと異なるならば、この送信イベントの直前にチェックポイントを設定し、一時停止する。
4. p_o はシステムメッセージ m_3 を Q の旧版プロセス q_o に送信する。m_3 には p_o がチェックポイントを設定しているか否かの情報が含まれる。

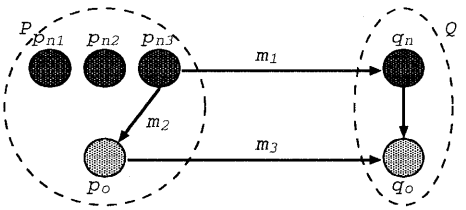

図 6: 多対一の改版における送信イベント

受信イベントにおけるプロセスの動作 (図 7)

1. アプリケーションメッセージ M を含むシステムメッセージ m_1 を受信した新版プロセス q_{ni}(ここでは q_{n1} とする) は、これをメッセージキューに登録する。また、M を含むシステムメッセージ m_2 を旧版プロセス q_o に対して送信する。
2. m_2 を受信した q_o は、この M が自プロセス内で発生した受信イベントで受理可能なメッセージであるならば M を受理する。そうでないならば、この受信イベントの直前にチェックポイントを設定し、一時停止する。
3. m_3 にチェックポイント設定済みの情報が含まれているならば、q_o は対応する受信イベントの直前にチェックポイントを設定し、一時停止する。

[3]ただし、q_n が未起動ならば q_o に対して送信する。

3.4 ロールバックプロトコル

過渡状態のシステムにおいては、改版過程クラスタに属する新版プロセスと未改版クラスタに属する旧版プロセスとの通信に未定義受信が発生する可能性がある。このとき、新版プロセスがアプリケーション実行を継続することは不可能であり、これを停止して旧版プロセスによる処理に切り換えるロールバック回復処理が必要である。あるクラスタにおいて未定義受信が発生した場合、未定義受信が発生したクラスタだけでロールバックを行なうのでは他のクラスタの状態との間の一貫性を保つことができない。

しかし、全てのクラスタでロールバックを行なう必要はない。例えば、ロールバックによって無効になるイベントと因果関係のないイベントのみを実行していたクラスタでは、ロールバック処理の必要はない。そこで、同時にロールバックを行なう必要のあるクラスタを定めるために、旧版プロセスはチェックポイント設定後に、チェックポイント設定情報を含むシステムメッセージを送受したクラスタの ID を CPL (チェックポイントリスト) に登録する。ロールバック回復処理の開始要求を伝えるために [9] のプロトコルを用いる。つまり、あるクラスタ C_i でロールバックを行なった場合には、C_i の CPL に ID が含まれる全てのクラスタでロールバック処理を行なうことで、一貫性の保たれたシステムの状態からの再実行が可能となる。クラスタ P 内で未定義受信が検出されたときの、ロールバック回復処理を以下に示す (図 8)。

1. 未定義受信を発生した新版プロセス p_n は、旧版プロセス p_o にシステムメッセージ m_1 を送信して、実行を終了する。
2. m_1 を受信した p_o は、CPL に ID が含まれる全てのクラスタの旧版プロセスにシステムメッセージ m_2 を送信する。
3. m_2 を受信した旧版プロセス q_o は、全ての新版プロセス q_{ni} に対してシステムメッセージ m_3 を送信するとともに、CPL に ID が含まれる全てのクラスタの旧版プロセスにシステムメッセージ m_5 を送信する。
4. m_3 を受信した q_{ni} は、q_o にシステムメッセージ m_4 を送信して実行を終了する。
5. p_o は、CPL に ID が含まれる全てのクラスタからシステムメッセージ m_5 を受信した時点でチェックポイントから処理を再開する。

この処理によって、ロールバックが必要なクラスタでロールバック処理が実行され、同時にプロセス間の通信チャネルを空にすることができる。

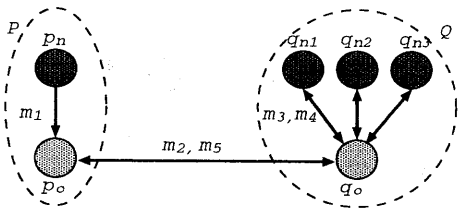

図 8: ロールバック回復処理

4 評価

本章では、3章で提案したプロトコルが、アプリケーションメッセージを過渡状態で伝達するのに要するシステムメッセージの数と所要時間を評価する。ここで、Δ はプロセス間のメッセージ伝達遅延とする。また、送信側クラスタ P と受信側クラスタ Q の構成により、以下の4通りが考えられる。

1. P は一つの新版プロセスと N_{so} 個の旧版プロセス、Q は一つの新版プロセスと N_{ro} 個の旧版プロセスで構成される。
2. P は一つの新版プロセスと N_{so} 個の旧版プロセス、Q は N_{rn} 個の新版プロセスと一つの旧版プロセスで構成される。
3. P は N_{sn} 個の新版プロセスと一つの旧版プロセス、Q は一つの新版プロセスと N_{ro} 個の旧版プロセスで構成される。
4. P は N_{sn} 個の新版プロセスと一つの旧版プロセス、Q は N_{rn} 個の新版プロセスと一つの旧版プロセスで構成される。

それぞれの場合に対して、最大、最小メッセージ数、所要時間を求めたものを表1に示す。

表 1: メッセージ数と所要時間の評価

	メッセージ数		所要時間	
	最大	最小	最大	最小
1	$2+3N_{so}+3N_{ro}$	$5+2N_{so}$	7Δ	6Δ
2	$4+3N_{sn}$	$4+2N_{sn}$	5Δ	4Δ
3	$2+3N_{ro}$	5	4Δ	4Δ
4	4	4	2Δ	2Δ

5 おわりに

従来の動的改版プロトコルでは、新旧版の対応関係が一対一に限定されていた。これに対して、本論文では、一対多、多対一の対応関係を持てるように拡張した新しいプロトコルを提案した。これによって、機能統合や機能分割をともなった動的改版を実現することができる。また、改版中に生じる未定義受信に対して、チェックポイントを用いて復旧する方法を示した。今後は、もう一つのプロトコルエラーである通信デッドロックに対処できるように提案プロトコルを拡張することが課題である。

参考文献

[1] Barbacci, M. R., Doubleday, D. L. and Weinstock, C. B. "Application Level Programming," *Proc. of 9th ICDCS*, pp. 458–465(1990).

[2] Chandy, K. M. and Lamport, L. "Distributed Snapshots: Determining Global States of Distributed Systems," *ACM Trans. Comp. Syst.* Vol. 3, No.1, pp. 63–75(1985).

[3] 桧垣,"分散システムにおける動的改版のためのグループ通信," 情報処理学会マルチメディア通信と分散処理ワークショップ, pp.81–90(1990).

[4] 桧垣, 平川, "分散システムにおける動的改版のためのグループ通信 (その 2)," 情報処理学会マルチメディア通信と分散処理研究会,94–DPS–65, pp.31–36(1993).

[5] 桧垣, 平川, "分散システムにおける動的改版のためのグループ通信 (その 3)," 情報処理学会マルチメディア通信と分散処理ワークショップ, pp.1–10(1994).

[6] 桧垣, 森保, 奥山, 平川, 市川, "システム進化支援環境リセプティブプラットホーム," 情報処理学会第 47 回全国大会, 6E–1, pp.201–202(1993).

[7] Kramer, J. and Magee, J., "Dynamic Configuration for Distributed systems," *IEEE Trans. on Software Eng.*, Vol. SE–11, No.4, pp. 424–436(1985).

[8] Kramer, J. and Magee, J., "The Evolving Philosophers Problem: Dynamic Change Management," *IEEE Trans. on Software Eng.* Vol. 16, No.11, pp. 1293–1306(1990).

[9] Moses, Y. and Roth, G. "On Reliable Message Diffunsion," *Proc. of 8th PODC*, pp. 119–127(1989).

[10] Segal, M. E. and Frieder, O. "Dynamically Program Updating in a Distributed Computer System," *IEEE Conf. on Software Maintenance*, pp. 198–203(1988).

[11] Shima, K., Higaki, H., and Takizawa, M., "Fault-Tolerant Causal Delivery in Group Communication," *Proc. of 10th ICPADS*, pp. 302–309(1996).

[12] Shiratori, N., Sugawara, K., Kinoshita, T. and Chakraborty, G., "Flexible Networks: Basic Concepts and Architecture," *IEICE Trans. on Communication*, Vol. E77–B, No.11, pp. 1287–1294(1994).

[13] 吉田 紀彦, "次世代並列分散システム開発のために," コンピュータ科学, Vol. 2, No. 4, pp. 300–305(1992).