システムソフトウェアと
オペレーティング・システム 65－6
（１９９４．７．２１）

超並列オペレーティングシステムにおけるプロセス移送問題

大澤 範高　　弓場 敏嗣

電気通信大学大学院
情報システム学研究科

超並列オペレーティングシステムにおけるプロセス移送の意義を示し、プロセス移送と耐故障性および楽観的実行との関連を述べる。プロセス間通信ネットワークを辿りながら移送先を順に決定するというアルゴリズムによる通信コストの改善効果の範囲を解析的に示す。改善効果をシミュレーションによって確認し、結果を考察する。

Process Migration on Massively Parallel Operating Systems

Noritaka OSAWA　　Toshitsugu YUBA

Graduate School of Information Systems
The University of Electro-Communications

1-5-1 Chofugaoka, Chofu, Tokyo 182, JAPAN

This paper explains the significance of process migration on massively parallel operating systems, and describes the relationship between process migration and other facilities, such as fault tolerance and optimistic execution. The authors propose a process migration algorithm which traverses inter-process connection network and determines the destination of a migration in order. Then the reduction of communication costs is analyzed. The result of the analysis is confirmed by simulation and the simulation studies are examined.

1 はじめに

近い将来、要素プロセッサ (PE) 数が 100 万規模の超並列計算機の実現が期待される。このような超並列計算機上で動作するオペレーティングシステム (OS) をここでは超並列オペレーティングシステム (超並列 OS) と呼ぶ。また、超並列計算機および超並列 OS などのシステムソフトウェアをまとめて超並列環境と呼ぶ。

超並列計算機のすべての PE を利用する場合に、複数のプロセスから構成されるジョブの実行が最速になるとは限らない。このため、PE 数が 100 万規模の超並列計算機は複数のジョブによって共有されることになり、システム内に複数のジョブが同時に存在することになる。

超並列環境では、固定的な数の PE を占有するのではなく、必要や環境に応じて PE の割り当ておよび解放を行なうジョブが増えると考えられる。本稿では、ジョブの処理進行に応じて PE 割り当てが変更される (プロセス数が変化する) として議論を進める。

1.1 負荷均衡と移送

必要時に PE の割り当てを連続的に行ない、不要時に PE の解放を行なうとすると使用されていない PE が多数発生することが予想される。ジョブ処理を繰り返すことに伴って平衡状態に近付くとし、各ジョブが要求する PE 数が同じになることはほとんどなく、ジョブの実行時間はそれぞれ独立で、割り当てが PE 番号の区間で行なわれる場合には、Knuth の 50%則によって、空き区間の平均数は、利用されている区間の半分になることがわかる。超並列計算機においては多数の PE が利用されないことになる。相互結合網のトポロジに応じた割り当ての方法をとることなどによって有効利用されない PE 数を少なくすることは可能であるが、より根本的に改善するためには、ジョブ実行中の移送が必要である。

また、ジョブを構成するプロセスのすべてを常に実行可能状態にすることはできず、待ち状態にあることもある。このような場合に、PE を有効に利用するために多重プログラミング機能によるプロセス多重化が必要である。プロセス多重化を行なう場合には、PE 間の負荷の均衡を図ることが、システムの性能向上のために重要である。負荷分散を適切に行なうためにも移送が必要である。

1.2 耐故障性と移送

超並列計算機においては、耐故障機能が不可欠である。PE あたりの平均故障間隔 (MTBF) を 10^7時間[1] とすると [1] 、100 万 PE の超並列計算機においては、MTBF は 10 時間となる。この MTBF よりも時間がかかる計算は、完了の見込みが少なくなる。超並列計算機は、大規模な計算の処理を目的としており、ジョブの計算時間が前記の MTBF よりも十分に小さいと仮定することは非現実的である。したがって、超並列計算機においては、故障の存在を前提とし、耐故障機能を備えることが不可欠となる。

耐故障機能を実現するためには、チェックポイントをとり、故障発生時に回復を行なえるようにする必要がある。このチェックポイントのためにデータ複製を行なうことは、移送の別の側面と考えることができる。したがって、耐故障性のためにデータ複製を行なうシステムにおいては、データ移動のための通信は、必ずしも移送のオーバヘッドとして考えなくてもよい。

超並列計算機では、すべての PE が 2 次記憶装置を持つことは難しい。したがって、PE の 1 次記憶領域にチェックポイントデータ (複製データ) が保持されると考える。また、すべてのチェックポイントを最初から 2 次記憶に保存する必要はなく、性能向上のために複数の版のチェックポイントデータが異なった記憶階層に保持されるようになると考える。より古い版が安定記憶に保持されているのであれば、最新のチェックポイントは他の PE の揮発性メモリに保持されていてもよい。それが必要に応じて、より安定な記憶に移される。

1.3 楽観的実行と移送

並列性をより引き出して利用するために、ある範囲の応用においては、緩い同期をとりながら実行を進め、不整合が発見された場合に回復を行なう楽観的実行が超並列環境において重要である。楽観的実行支援は、超並列 OS が備えるべき機能の一つであ

[1] ほとんどのハードウェアモジュールの MTBF は、1985 年に 1 年程度であったが 1990 年には 10 倍程度になったと [1] に記述がある。そこで、ここではさらなる改善を見込んで、100 年 $\approx 10^7$ 時間 とした。

ると考える。楽観的実行を支援する場合には、状態の回復機能が必要である。回復のためには、耐故障性の場合と同様に状態保存が必要であり、そのためにデータ移動 (複製) が必要である。楽観的実行支援機能を持ったOSにおいても、移送のデータ移動は、必ずしも移送のオーバヘッドとして考えなくてもよい。

1.4 従来の研究

従来の分散システムのプロセス移送研究における移送の評価基準には、応答時間 (ターンアラウンド時間) が広く利用されている [2] [3]。超並列計算機の利用は、処理時間短縮を目的とするものであり、移送による応答時間の短縮は必要である。しかし、超並列計算機でも汎用を目指すならばスループットの向上を図ることも必要である。本稿では、先に示したようにデータ移動を移送の有無にかかわらず行なわなければならない超並列環境において、スループット性能が移送によっていかに向上するかを検討する。

2 移送の評価要因

ある期間におけるコストとメリットの関係について検討する。移送のコストは、プロセス移送に必要な平均コスト α と期間あたりの移送すべきプロセス数 N の積で表すことができる。一方、超並列環境において、移送によって生まれるメリットは2種類に大別できる。局所性を高めることによって通信コストを低減できることと負荷分散によって実効処理能力を高められることである。これは、プロセスあたりの低減できる平均通信コスト β と負荷分散による平均処理能力向上 γ として表すことができる。すなわち、移送によって性能が向上するためには、次の関係が成り立つ必要がある。M を移送のメリットを受ける期間内に生成されるプロセス数とする。

$$\alpha N \leq \beta M + \gamma M$$

左辺は移送のコストであり、右辺は移送によって得られるメリットである。左辺と右辺の差が最大になるようにすることが重要である。

超並列環境においては、α がデータ移送のコストを必ずしも含まないことは前述の通りである。βは、移送後の配置をどのようにするかによって影響を受ける。γは、完全に負荷分散できた場合にどの程度近付くことができるかが問題である。M は、期間がきまればプログラムに依存して決まる。ただし、期間を長くすることによって大きくすることが可能である。本稿では、β についての解析を行なう。N、γ の問題については世代的移送の節で触れる。

3 通信コストの低減

ジョブを構成するプロセスの地理的局所性を高めることによる平均通信コストβの低減を検討する。

3.1 移送アルゴリズム

まず、例を使って移送アルゴリズムを説明する。図 1のような2分木の相互結合網トポロジを持った並列計算機を考える。この並列計算機上でプロセスA、B、C、D が動作しており、それらのプロセスは図のようにPEに割り当てられているとする。PE間の実線で示された線分の本数が、距離を表すとする。また、破線で結ばれたプロセス (PE) 間で通信が行なわれており、プロセス間の通信量はすべて1とする。この場合に、システム全体の通信総コストは、12 である。

ここでは、A から始めて、プロセス間通信グラフ (ネットワーク) を辿ってその順に移送を行なう。このアルゴリズムが適用されると図 2のように配置される。通信総コストは減少し、10 となる。このようにして、移送によってシステム全体の性能の向上を図ることが可能である。図のBとCとの距離の変化からもわかるように、このアルゴリズムでは、通信コストが常に小さくなることは保証されない。

例では、移送先の決定 (移送起動処理) とデータ移動が同時に行なわれるとして説明したが、実際には、移送起動処理とデータ移動は分離して処理して良い。ここで規定するのは移送起動処理についてである。

各プロセス間通信は、コネクションベースで行なわれるとする。コネクションレスの場合には、通信の度に、コネクションの設定、通信、コネクションの切断が行なわれるとすれば、このアルゴリズムを適用できる。

システムには、移送管理プロセスが存在し、ある時点でのスナップショットにおけるすべてのプロセスは、移送管理プロセスからコネクションを辿ること

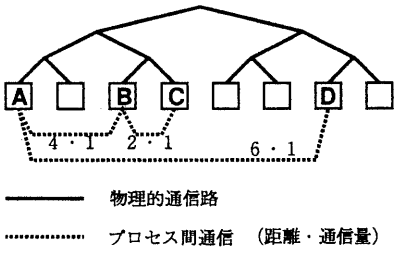

図 1: 移送前の状態 – 破線の近くの 4・1 の前の数値が距離を表し、後ろの数値が通信量を表す。

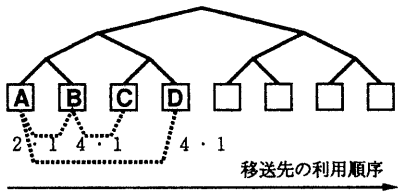

図 2: 移送後の状態

によって到達可能とする。移送管理プロセスが移送開始タイミングを判断する。この判断は、タイマーや負荷状況に基づいて行なわれるが、移送開始タイミングは、ここで扱うアルゴリズムとは分離する。ここで規定するのは、移送先の決定手順である。データを実際に移送する機構とシステム全体の移送完了検出部分は含まない。また、移送先の利用順序は、トポロジおよび利用形態に応じて最適になるようにあらかじめ決める必要がある。利用順序の優劣を決める基準については、後の節で検討する。

プロセスの移送処理の状態遷移図を図3に示す。また、動作を C++ 言語風に記述したものを付録 A に示す。

各プロセスの移送関連状態は、非移送状態、移送起動処理中、移送起動処理完了に分けられる。初期状態は非移送状態である。各プロセスには、接続、切断、移送起動指示、移送起動終了、リセットという事象が発生する。接続、切断は、それぞれ、プロセス間のコネクションの接続、切断時に発生する事象である。移送起動指示は、プロセスの移送起動処理を指示するメッセージによって発生する。また、移

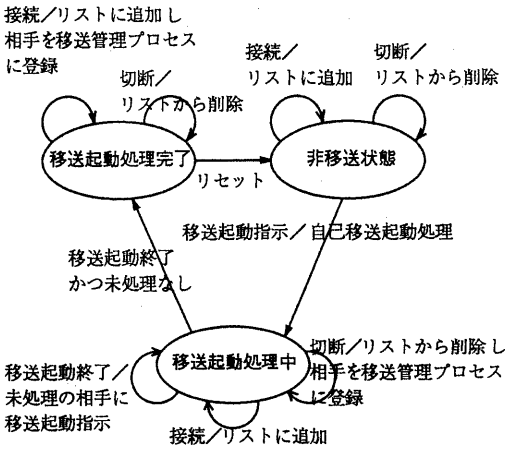

図 3: 状態遷移図 – リストとは、通信相手を管理するリストを意味する。

送起動終了は、プロセスの移送起動処理が終了した際に、移送起動指示を行なったプロセスに発生する事象であり、移送そのものの完了を意味しない。リセットは、データの移動を含めて移送が完了した場合にシステムから与えられる事象である。

3.2 アルゴリズムの利点、欠点

ここで示したアルゴリズムの利点は、次の通りである。(1) プロセス間通信グラフを辿りながら移送起動を行なうことによって、個々の PE に処理を分散させることができる。(2) あらかじめ移送先の利用順序を決めておくので、負荷等に応じて最適な配置を移送処理時に計算する場合に比べて、実行時オーバーヘッドが少ない。(3) プロセスの状態を増やし、「色」をつければ、移送処理を多重化させて進めることができる。(4) 移送先の順序が決っているので、同じプロセスの移送が繰り返される場合に発生する周期の短い振動現象が起きない。

欠点は、次の通りである。(1) 移送起動指示を受け取ってから移送起動終了を送信するまでに、プロセスに故障が起きると処理が停止する。しかし、各プロセスが移送先を移送管理プロセスと通信して決定するようにすれば、移送管理プロセスがタイムアウトによって故障を検出することが可能である。(2)

あらかじめ移送先の利用順序を決めておくので、最適の配置にはならない。

4 移送の効果

移送による通信コストの低減効果の範囲を解析的に検討する。これによって、移送先の利用順序の評価を行なうことができる。

まず、プロセスの全体数を N とし、その間の ${}_N\mathrm{C}_2$ 個のリンクを考える。プロセス i からプロセス j へのリンクの距離、通信量をそれぞれ d_{ij}、m_{ij} とする。

システム全体の通信総コストを

$$C = \sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}m_{ij}$$

と表す。

移送を行なわない場合の通信総コストを C_{nomig} とし、移送を行なった場合の通信総コストを C_{mig} とする。通信総コスト改善率 Bを

$$B = \frac{C_{nomig} - C_{mig}}{C_{nomig}}$$

と定義する。

4.1 条件

通信量の変化はマルコフ過程とし、すべてのプロセス間通信の統計的な性質は同じとする。すべてのプロセス間での通信量を計算することによって、複数のジョブとジョブ内プロセス並列度変化およびプロセス間通信量変化を近似する。

最も基本的な場合として、$d_{ij} = d_{ji}$、$m_{ij} = m_{ji}$ で、通信量 m_{ij} が 0 または 単位量 1 のいずれかである場合を検討する。単位時間あたりの通信量の種類を増やし、通信量を考慮した場合には、2 種類の場合よりも良い改善率が得られると考えられる。

通信量 0 から 0 への推移確率 p_{00} と通信量 1 から 1 への推移確率 p_{11} が解析のパラメータとなる。

4.2 活動率と平均通信相手数

通信が行なわれているリンクを活動リンク (コネクション) と呼ぶ。定常状態において活動リンクのリンク全体に占める割合 (活動リンク率) は、$\overline{r_{a_l}} = (1-p_{00})/(2-p_{00}-p_{11})$ であり、平均活動リンク数は、$\overline{n_{a_l}} = N\,\overline{r_{a_l}}$ となる。

次に、定常状態においてプロセスが通信する相手の平均数である平均通信相手数 $\overline{n_p}$ を求める。少なくとも一つのリンクで通信が行なわれているプロセスを活動プロセスと呼び、定常状態において、活動プロセスのプロセス全体に占める割合を活動プロセス率 $\overline{r_{a_p}} = 1-(1-\overline{r_{a_l}})^N$ とする。また、定常状態における活動プロセスの平均数を平均活動プロセス数 $\overline{n_{a_p}} = N\,\overline{r_{a_p}}$ とする。この場合、$\overline{n_p} = \overline{n_{a_l}}/\overline{r_{a_p}}$ となる。

4.3 近傍平均距離と改善率

あるトポロジにおいて、移送先として利用する順序に番号をノードにふる。番号の小さい方から n 個のノードにおいて、近い番号の m ノード間の平均距離を近傍平均距離 $L(n,m)$ とする。

$$L(n,m) = \frac{1}{s(n,m)}\sum_{i=1}^{n}\sum_{j=i+1}^{min(i+m-1,n)} d_{ij}$$

ここで、$s(n,m)$ は、n ノード中の近傍 m ノード間のリンクの数である。

通信総コスト改善率 Bは、全ノード間の平均距離 $\overline{d} = L(N,N)$ に対する実効的な平均距離の改善率によって表せる。実効的な平均距離の上限および下限は、近傍平均距離を基に表せると考える。その場合に、近傍平均距離のパラメータ n は、$\overline{n_{a_p}}$ になり、m は、$\overline{n_{a_l}}$ から $\overline{n_{a_p}}$ の間と考えられる。ただし、通信をしているプロセスは移送によって近傍に集まるが、推移確率に応じて通信相手が変化するので、移送後のプロセス間の実効的な近傍平均距離は、

$$l(n,m) = p_{11}\,L(n,m) + (1-p_{11})\,\overline{d}$$

となる。

そこで、平均距離実効改善率 $E(n,m)$ を

$$E(n,m) = \frac{\overline{d} - l(n,m)}{\overline{d}}$$

とすると、実際の改善率は、以下の関係にあると考えられる。

$$E(\overline{n_{a_p}}, \overline{n_{a_p}}) \le B \le E(\overline{n_{a_p}}, \overline{n_{a_l}})$$

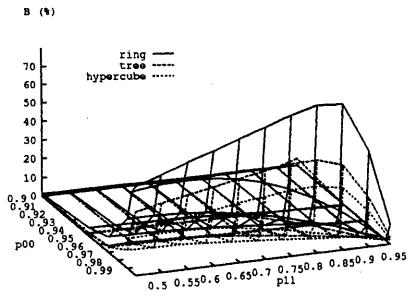

図 4: 128 台の場合の改善率 (B) – ring、tree、hypercube は、それぞれ、リング、2分木、ハイパーキューブを意味する。

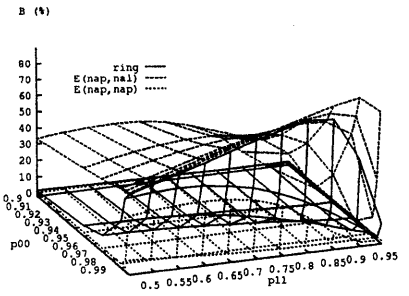

図 5: 128 台リング構造の改善率 (B) と平均距離実効改善率 – ring は、リング構造のシミュレーション結果を表す。E(nap,nal)、E(nap,nap) は、それぞれリング構造の場合の $E(\overline{n_{a_p}}, \overline{n_{a_l}})$、$E(\overline{n_{a_p}}, \overline{n_{a_p}})$ を意味する。

5 シミュレーション

前記のアルゴリズムによるシステム全体の通信コストの低減の解析結果をシミュレーションによって確かめた。通信量の推移確率パラメータを変化させてシミュレーションを行なった。

シミュレーションの期間は、1000単位時間とした。1単位時間には、前記のアルゴリズムによる一連の移送処理が行なわれる。本稿の通信量の推移確率行列は、正則なチェーンの推移確率行列となっている。回数を i とすると初期値の影響は、$(p_{00}+p_{11}-1)^i$ に従って減少する。以下のシミュレーションでは、$p_{11} < 0.95$ であり、1000 回の繰り返しで、5.29×10^{-23} 以下となる。したがって、本シミュレーションの条件において、初期値は結果に影響を与えない。

5.1 シミュレーション結果

シミュレーション結果を図 4 に示す。リング、2分木、ハイパーキューブに対するシミュレーション結果を示している。図5に、リング構造で PE が 128 台の場合の改善率と平均距離実効改善率を示す。図 6 は、図 4と図 5 の格子点を平均通信相手数と改善率との関係でプロットしなおしたものである。

前節の解析および本節のシミュレーションから導けることを以下にまとめる。

- リング構造が最も効果が高く、2分木、ハイパーキューブの順に効果が少なくなる。
- リング構造において解析的に考えた改善率の範囲にシミュレーション結果が入っている (図 5)。前節の解析は、特に要素数に依存しないので、超並列においても同様の議論が成り立つ。
- 近傍平均距離に基づいた平均距離実効改善率によって、移送先 PE の利用順序を評価できる。
- 平均通信相手数が少ない場合には $E(\overline{n_{a_p}}, \overline{n_{a_l}})$ に近いが、増えるに従って $E(\overline{n_{a_p}}, \overline{n_{a_p}})$ に近付く (図 6)。これは、通信相手数が増えるに従って通信相手を連続に配置することができなくなり、近傍平均距離における近傍の範囲が上限に近付くためと考えられる。

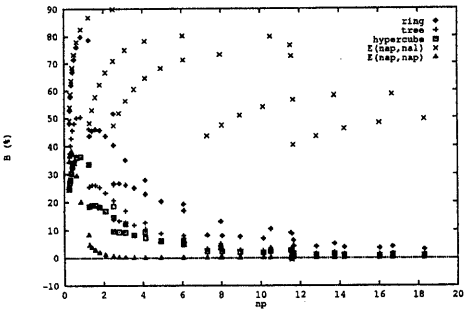

図 6: 128 台の場合の平均通信相手数と改善率の関係 – 図 4と図 5 の格子点を平均通信相手数 (np) と改善率 (B) との関係でプロットしなおしたもの。

- プロセス間通信のグラフが木構造をとる場合の平均通信相手数は2に近い。その場合には、本稿の簡単なアルゴリズムでも 10%～30% 程度の改善が得られる。平均通信相手数が大きい場合にはほとんど改善が見られない。しかし、入出力数 (fan-in, fan-out) の多いスイッチから構成される相互結合網を性能価格比良く構築することは難しい。したがって、超並列計算機において平均通信相手数が大きい場合には、移送方式によらず通信コストの低減は難しいと思われる。

6 世代的移送

超並列計算機においては、プロセス総数は多い。前記のアルゴリズムのまま移送起動処理を順次行なうとすると、移送起動の時間間隔が長くなり過ぎて移送の効果を十分に引き出せないと考えられる。このため、対象のプロセスを分割し、並行して処理することが必要である。

また、すべてのプロセスを平等に移送することは、無駄が多い。必要に応じて移送の頻度を調整できるようにすべきである。移送を行なうべき頻度に応じて対象のプロセスを分割することによって、この目的を達成することが可能である。

6.1 領域分割

分割の基準としては、プロセスの空間的配置とプロセスの生存時間が考えられる。

プロセスの空間的配置に基づく分割は、分割された部分単位内で移送を行なうものである。並行して処理を行なうということは可能であるが、初期配置によって移送の範囲が決定されてしまう。

プロセスの生存時間に基づく分割は、生存時間ごとに異なる領域へ移送を行なうものである。各領域に存在するプロセスの生存時間は同程度となる。

6.2 生存時間と移送効果

移送の効果を高めるためには、寿命 (生成から消滅までの時間) の長いプロセスに対してのみ移送を行なうことが重要である。なぜなら、寿命の短いプロセスは、移送処理中に寿命に達することが多く、移送が無駄になる可能性が高いからである。一方、寿命の長いプロセスの場合には、移送の効果がある可能性が高くなる。したがって、プロセスの寿命の予測を行ない、その予測精度を高めることが必要である。

超並列計算機システムにおけるプロセスの寿命分布については十分なデータがない。しかし、テスト、デバッグなどの際の比較的短命なプロセスは多数あり、本格的な大規模計算のプロセスは少ないことが予想される。このことから、超並列計算機におけるプロセスの寿命も言語処理系などにおけるオブジェクト (もしくは、セル) の寿命 [4] と同様の性質を持つと考えられる。

世代型ガーベジコレクション [4] では、生存時間の短い区分ほど多数回の処理を行なう。一方、移送では、データの生存時間が短い場合には初期状態からの再実行による回復が可能であり、移送を行なっても短時間でプロセスが終了する可能性が高いのであまり頻繁に行なわない方が良いと考えられる。また、生存時間が十分に長いものは最適な配置に近くなっているので移送はあまり必要がなくなると思われる。

7 負荷均衡による性能向上

プロセス数が PE 台数よりも多い場合には、負荷を均衡させることによって性能向上が期待できる。ここで示したアルゴリズムはプロセスをあらかじめ定めた順序に従った移送先に移すので、PE 当たりのプロセス数を均衡に近付くように移送できる。

プロセス数が多くなった場合には移送処理に時間がかかり、プロセスの寿命の違いによる負荷の不均衡が発生する。しかし、これもプロセスを生存時間に基づいて世代に分割して管理することによって、均衡を図ることができると考える。

8 おわりに

本稿では、プロセス間通信グラフを辿りながら移送先を順に決定するという簡単なアルゴリズムによって通信コスト改善が図れることを示した。

世代分割による性能向上や負荷分散の効果については今後詳しく検討する予定である。また、本稿では、移送の機構については触れなかったが、移送、耐故障機能、楽観的実行支援の基礎となる超並列計算

機向きの複数の版をサポートするチェックポイント機構を研究する予定である。

参考文献

[1] Gray, Jim, "Census of Tandem System Availability Between 1985 and 1990," IEEE Trans. on Reliability, Vol.39, No.4, pp.409-418, Oct. 1990.

[2] Shivaratri, Niranjan G. et al., "Load Distributing for Locally Distributed Systems," IEEE Computer, Vol.25, No.12, pp.33-44, Dec. 1992.

[3] 朴 圭成, 芦原 評, 清水 謙多郎, 前川 守, 「分散オペレーティングシステムにおけるプロセス移送の方式」, 情報処理学会論文誌, Vol.31, No.7, pp.1080-1090, July 1990.

[4] Lieberman, H. and Hewitt, C., "A Real-Time Garbage Collector Based on the Lifetimes of Objects," CACM Vol.26, No.6, pp.419-429, June 1983.

A 移送アルゴリズム概要

ここに示したプログラムでは、例として preorder でグラフを辿り移送起動処理を行なうとした。異なる順序で移送起動処理をすることも可能である。また、各プロセスで次の移送先を計算し、その情報を事象の引数として渡すとした。すなわち 移送起動指示と共に次に利用すべき移送先情報が渡され、移送起動終了と共に更新された移送先情報が渡される。通信が必要になるが、移送管理プロセスに問い合わせながら移送先を決定することも可能である。

InitiateSelfMigration は、自己プロセスの移送起動処理を始めることを意味する。すなわち、あらかじめ定められた順序に従って移送先を決め、そこへのデータ移動の準備等を行なう。SendEvent は、第一引数で指定されるプロセスに対して、事象を送信することを意味する。SelectNext は、次の移送起動対象を選択する。すべての移送起動処理が終了している場合には 0 を返すとする。

```
Process(Event event, Node from, Node next)
{
    extern Node   Manager;  // 移送管理プロセス
    static State  state = NotMigrated;
    Node          parent, to;
    List          peers;    // 現在の通信相手
    List          toMove;   // 移送起動処理すべき相手

    switch ( event ) {
      case Connected:         // 接続
        peers.append(from); // リストに追加
        break;
      case Disconnected:      // 切断
        peers.remove(from); // リストから削除
        break;
    }
    switch ( state ) {
      case NotMigrated: // 未移送
        switch ( event ) {
          case Initiate:      // 移送起動指示
            parent = from;
            toMove = peers;
            state = InProgress; // 移送中
            InitiateSelfMigration(next++);
            SendEvent(self, Finished, next)
            break;
        }
        break;
      case InProgress:  // 移送起動処理中
        switch ( event ) {
          case Finished:      // 移送起動終了
            toMove.remove(from);
            to = SelectNext(toMove); // 次を選択
            if ( to ) { // 移送を指示
                SendEvent(to, Initiate, next);
            } else {
                state = Completed; // 移送起動処理終了
                SendEvent(parent, Finished, next);
            }
            break;
          case Disonnected: // 切断
            // 移送管理プロセスに登録
            SendEvent(Manager, Connected, from);
            break;
        }
        break;
      case Completed:   // 移送起動処理完了
        switch ( event ) {
          case Connected:     // 接続
            // 移送管理プロセスに登録
            SendEvent(Manager, Connected, from);
            break;
          case Reset:         // リセット
            state = NotMigrated; // 非移送状態
            break;
        }
        break;
    }
}
```