アルゴリズム 47－4
（1995．9．21）

分散移動システムにおけるスナップショット・アルゴリズム

佐藤 泰朗　井上 美智子　増澤 利光　藤原 秀雄

奈良先端科学技術大学院大学 情報科学研究科
〒630-01 奈良県生駒市高山町 8916-5

e-mail: yasuro-s@is.aist-nara.ac.jp

移動計算機を含む分散システム（分散移動システム）のための分散アルゴリズムについて考察する．これまでに提案されている分散アルゴリズムのほとんどは，移動計算機を含まない分散システムを対象としており，分散移動システムに適用できない．

本稿では，まず，分散アルゴリズム設計のための分散移動システムのモデルを提案する．次に，分散システムにおいて，基本的，かつ，重要な問題の1つであるスナップショット問題を，分散移動システムで定義する．分散移動システムでは，移動計算機の移動に伴い，ネットワークの形状が変化するので，この定義では，ネットワーク形状の無矛盾性も考慮している．そして，分散移動システムでのスナップショットアルゴリズムを示す．

A Snapshot Algorithm for Distributed Mobile Systems

Yasuro Sato, Michiko Inoue, Toshimitsu Masuzawa and Hideo Fujiwara

Graduate School of Information Science
Nara Institute of Science and Technology (NAIST)
8916-5 Takayama, Ikoma, Nara, 630-01 Japan

e-mail: yasuro-s@is.aist-nara.ac.jp

This paper considers distributed algorithms for mobile systems. Many distributed algorithms have been designed for distributed systems consisting of static hosts. But most of them cannot be used for mobile systems.

In this paper, we propose a model of mobile systems for design of distributed algorithms. We give a definition of a snapshot problem, one of the fundamental problems, in the model. Since the network topology of a mobile system changes, we consider topological consistency in the definition. We also present a snapshot algorithm for mobile systems.

1 まえがき

移動／携帯計算機の普及, 及び, 無線通信の発達に伴い, 固定計算機と移動計算機が混在する分散移動システムの重要性が高まってきている. そのため, 分散移動システムについて多くの研究が行われている [1][5]. 分散移動システムは, 従来の計算機ネットワークに移動計算機を付加したものであり, 固定計算機は（静的な）チャネルで相互結合されている. 移動計算機とは, 計算機ネットワークに接続したまま移動可能な計算機のことである. 固定計算機の中には, いくつかの移動支援局が含まれている. 各移動支援局は, 地理的あるいは論理的な支援領域を持ち, その支援領域内に存在する移動計算機と無線で相互通信可能である. 移動計算機が, ある移動支援局の支援領域から他の移動支援局の支援領域に移動すると, 通信可能な移動支援局が変化する. そのため, 移動計算機と移動支援局との間のチャネルは, 移動計算機の移動にともない接続関係が変化する.

分散移動システムでは, システム全体で何らかの処理を実行中に移動計算機が他の支援領域に移動した場合にも, 処理を続けられることが望まれる. 分散システムで問題を解くためのアルゴリズムは分散アルゴリズムと呼ばれ, これまでに多くの研究が行われている [7][12]. しかし, これらのアルゴリズムは, 移動計算機を含まない従来の分散システムを対象としており, 接続関係が動的に変化する分散移動システムでは動作しない. そのため, 分散移動システムを対象とした分散アルゴリズムの設計が必要であり, いくつかの研究が行われている [1][2]. しかし, 分散移動システム上の分散アルゴリズムに関する研究はまだまだ不十分であり, 分散アルゴリズム設計のための分散移動システムのモデルも確立していないのが現状である.

そこで本稿では, 分散アルゴリズムの設計のための分散移動システムのモデルを提案し, スナップショット問題という基本的, かつ, 重要な問題について, 分散移動システム上のアルゴリズムを提案する.

移動計算機の移動により, その移動支援局が変化することを考える. この場合, 実行中の分散アルゴリズムを継続実行するために, 移動前の移動支援局から, 移動後の移動支援局に何らかの情報を転送しなければならないことが多い. そこで, 分散移動システムでは, この情報の転送のために移動管理手続き（*handoff procedure*）と呼ばれる手続きが用意されており, 移動計算機が移動支援局を変更する際に実行される [1]. そこで, 本稿で提案する分散移動システムのモデルでは, この移動管理手続きをシステムの一部と考え, 分散アルゴリズムで利用できる移動管理手続きの機能をモデルの一部として定義している. そして, 移動計算機の移動を, その際に実行される移動管理手続きが引き起こすイベントとしてモデル化している. ただし, 現状では, 実際の移動分散システムで用意される移動管理手続きも確定していないので, 移動管理手続きの機能としては, 基本的なものに限っている. 紙面の都合で, 移動管理手続きの実現については触れないが, 本稿のモデルで定義する移動管理手続きは, 簡単に実現できるものである.

本稿では, 分散移動システムでのスナップショット問題を考える. スナップショット問題とは, 分散アルゴリズムの実行に対して, システム全体の状況を求める問題である. この求められた状況をスナップショットという. スナップショットは, アルゴリズムの終了, デッドロック, トークンの紛失などの定常特性の検出／判定に利用される [4][9]. また, 故障から復帰するためのロールバック点 [6][11] や分散プログラムのデバッグ [3][10] にも利用されている. このように, スナップショット問題は, 他の多くの問題を解くのに利用される基本的な問題である.

スナップショットは, システム全体の状況であるので, システム内のすべての計算機の状態を含んでいる. しかし, 分散システムには一般に大域時計が存在しないので, すべての計算機を同時に観察することはできない. そのため, それぞれ異なる時刻に観察したホストの状態が, 意味を持つスナップショット, すなわち, 無矛盾なスナップショットを構成することが必要である.

無矛盾なスナップショットを求める方法は, 大きく２つに分類できる. 一つは, 各移動計算機が独立に定期的にその状態を記録しておき, それらの状態から適当なものを集めることにより無矛盾なスナップショットを構成するという方法である. この方法は, 各計算機が独立に状態を記録することから, 非同期式スナップショットアルゴリズムとよばれる. 非同期式スナップショットアルゴリズムは, ロールバック点を求める方法としては, よく利用されるが, 各計算機がいくつもの状態を記録するために, 多くのメモリを必要とする. これに対し, システム内のすべての計算機が協調し, 各計算機が無矛盾なスナップショットを構成するために１つの状態だけを記録する方法は, 同期式スナップショットと呼ばれる. 移動計算機の存在しない分散システムでは, Chandy と Lamport により, 効率のよい同期式のスナップショットが提案されている [4].

分散移動システムでのスナップショットアルゴリズムとしては, 故障から復帰するためのロールバック点として, スナップショットを求める非同期式スナップショットアルゴリズムが提案されている [2]. しかし, スナップショットの対象となる分散アルゴリズムについて, 移動計算機でのみ動作し, 移動計算機間には静的な論理チャネルが実現されていると仮定している. そのため, 接続関係が動的に変化するモデルには適用できない.

そこで, 本稿では, 移動計算機の移動により接続関係が変化する分散移動システムにおける, 同期式スナップショットアルゴリズムを提案する. 本稿では, 分散移動システムでのスナップショットを考えるにあたり, 無矛盾な状況に, さらに, ネットワーク形状の無矛盾性という条件を付加した強無矛盾な状況を定義する. そして, 分散移動システムでのスナップショットアルゴリズムを強無矛盾な状況を求めるアルゴリズムと定義する.

２節では, 分散移動システムのモデルを示し, 分散移動システム上でのイベント, 無矛盾な状況, 強無矛盾な状況などを定義する. ３節では, 分散移動システムにおけるスナップショットアルゴリズムを定義し, 実際に設計したアルゴリズムを紹介する.

2 分散移動システムのモデルと定義

分散移動システム Sを３項組 S=(MSS,MH,CH) と定義する. ここで, MSS は移動支援局 (*mobile support station*, 以下 MSS) と呼ばれる固定計算機の集合, MH は

移動計算機 (*mobile host*, 以下 MH) の集合, **CH** は MSS 間の静的チャネル (*static channel*) の集合 (すなわち,**CH** は, 相異なる 2 つの MSS の非順序対の集合) である. **MSS** と **CH** で形成されるネットワークを Sの MSS ネットワークと呼ぶ. MSS ネットワークは連結であると仮定する. S の 3 項組では MH と接続するチャネルを明示していないが, 各 MH は高々 1 つの MSS と動的チャネルによって接続している. 動的チャネルは, 無線チャネルなどで実現され,MH の移動によりその接続する MSS が変化する. そのため, 動的チャネルを MH の状態で表すことにする (後述). 本稿では,MH の移動は動的チャネルの変化としてモデル化する.

分散移動システム S=(**MSS**,**MH**,**CH**) に対し, $\mathbf{P} = \mathbf{MSS} \cup \mathbf{MH} = \{p_1, \ldots, p_N\}$ とし, **P** の要素をホストと呼ぶ. ここで,Nはホストの個数である. 各ホストは識別子を持つ. 簡単のため, ホスト p_iの識別子 $ID(p_i)$ を, 単に,p_iと表す. ホスト p_iとホスト p_jの間にチャネルが存在するとき, ホスト p_jをホスト p_iの隣接ホストという. 特に,p_jが MSS のときは, 隣接 MSS という. 隣接 MH についても同様に定義する. MH 間にチャネルは存在せず, MH 間の通信は MSS ネットワークを経由してのみ可能である.

ホストを状態機械としてモデル化する. ホストは, 隣接ホストの識別子を状態の一部に保持しており, ホストの状態 c は,$c = (I, L)$ と表される. ここで,L は隣接ホストの識別子の集合, Iはそれ以外の状態を表す. ただし,MH は高々 1 つの動的チャネルに接続するので, MH の L は要素数が高々 1 の集合である. 以下では, ホスト状態 $c = (I, L)$ に対し,Iを内部状態, L を接続状態と呼ぶ.

すべての静的/動的チャネルは双方向チャネルであり, それぞれの方向について長さに制限のない FIFO キューとしてモデル化する. MSS p_iと MSS p_jの間に静的チャネルが存在するとき, p_iから p_jへの FIFO キューを $q(p_i, p_j)$, p_jから p_iへの FIFO キューを $q(p_j, p_i)$ と表す. また, 動的チャネルについては, 接続関係が変化するが, 各 MH は高々 1 つの動的チャネルに接続するので, 各 MH p_iに対し, 2 つのキュー $q^-(p_i)$,$q^+(p_i)$ を用意する. $q^-(p_i)$ は,MH p_iが隣接 MSS からメッセージを受信するキュー, $q^+(p_i)$ は,MH p_iが隣接 MSS へメッセージを送信するキューを表す. 静的チャネルの個数を M_s,MH の個数を M_mとすると, $M = 2(M_s + M_m)$ 個の FIFO キューが存在し, これらを $q_1, \ldots, q_M$と表す. 図 1に分散移動システムの例を示す.

分散移動システムにおいては, MH の移動による動的チャネルの付け替えは, 移動管理手続き (*handoff procedure*) によって行なわれる [1]. 移動管理手続きは, MH が移動して隣接 MSS が変化するときに, 移動に関わるホスト間で情報を交換する. 本稿では, 以下のような移動管理手続きを仮定する. MH p_iが移動して, 隣接 MSS が p_jから p_kに変わる場合, 次の順序で接続状態が変化する.

(1) p_iの接続状態 L_iが空集合になる.

(2) p_jの接続状態 L_jから p_iが削除される.

(3) p_kの接続状態 L_kに p_iが追加される.

(4) p_iの接続状態 L_iが $\{p_k\}$ になる.

移動管理手続きでは, 上記の順に接続状態の変化を起こすために, p_iから p_j,p_jから p_k,p_kから p_iの順に何らかのメッセージが送信されると考えられる. そこで, これらのメッセージを利用して,p_i,p_j,p_kの間で情報を交換できると仮定する. このような情報を移動情報と呼ぶ. 移動情報を新たにシステム内に流すことを, 移動情報を追加する, 移動情報を受け取ることを, 移動情報を取得するという.

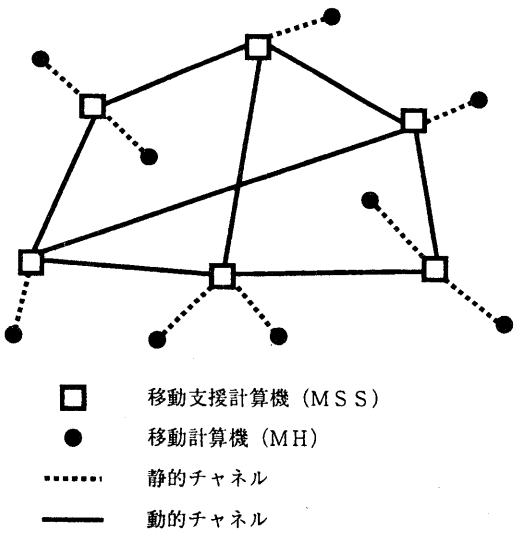

図 1: 分散移動システム

分散移動システムの大域状況 γ は, システム内のすべてのホストの状態, すべてのチャネルのすべての FIFO キューの状態 (すなわち, 伝送中のメッセージの系列), システム内に存在する移動情報からなる. ホスト $p_i, \ldots, p_N$の状態を $c_1, \ldots, c_N$ (ただし,$c_i = (I_i, L_i)$), チャネルのキュー $q_1, \ldots, q_M$の状態を $d_1, \ldots, d_M$, システム内に存在する移動情報の集合を Fと表すと, 分散移動システムの状況 γ は,$\gamma = (c_1, \cdots, c_N, d_1, \ldots, d_M, F)$ と表される. 特に, 以下の 5 つの条件を満たす状況 $\gamma = (c_1, \ldots, c_N, d_1, \ldots, d_M, F)$ を初期状況と呼ぶ.

(1) $I_1, \ldots, I_N$は, それぞれ $p_1, \ldots, p_N$の初期内部状態.

(2) 任意の MH p_iの接続状態に, ある MSS が属する.
すなわち,$\forall i\ [p_i \in \mathbf{MH} \Longrightarrow L_i \neq \emptyset]$.

(3) ホスト p_iがホスト p_jの接続状態 L_jに属するとき, ホスト p_jはホスト p_iの接続状態 L_iに属する.
すなわち,$\forall i, j\ [p_i \in L_j \Longleftrightarrow p_j \in L_i]$.

(4) すべてのチャネルにはメッセージが存在しない. すなわち, $\forall i[d_i = \varepsilon]$ (εは空系列を表す).

(5) システム内に移動情報は存在しない. すなわち, $F = \emptyset$.

ホストでイベントが生起することにより, 分散移動システムの状況が変化する. 次に, 分散移動システムにおけるイベントを定義する. ホストの移動を伴わない従来の分散システムのモデルで考えられている, 内部イベント, 送信

イベント, 受信イベントの3種類のイベント [12] に加え, MHの移動を表すために, 新たに4種類のイベントを導入する. これらは, 前述の接続状態が変化する時に起こるイベントで, 生起する順に, 切断イベント, 支援終了イベント, 支援開始イベント, 接続イベントと呼ぶ. また, この4種類のイベントを移動イベントと呼ぶ.

定義 1 イベント
分散移動システムの7種類のイベントを定義する.
以下, 各イベントについて,

(a) 概要の説明

(b) 引数の条件

(c) イベントが状況 $\gamma = (c_1 = (I_1, L_1), \ldots, c_N = (I_N, L_N), d_1, \ldots, d_M, F)$ で生起可能である条件

(d) イベント生起後の状況$\gamma' = (c'_1 = (I'_1, L'_1), \ldots, c'_N = (I'_N, L'_N), d'_1, \ldots, d'_M, F')$ について, γから変化する部分

を示す. 以下では,$c = (I, L)$,$c' = (I', L')$ はホストの状態を表す. 簡単のため, 各メッセージ m は, 送信ホストの識別子 $(src(m))$ と宛先ホストの識別子 $(dst(m))$ を含んでいるものとする. また,$p_i \in \mathbf{MSS}$ なら, $MH\ p_j \in L_i$のとき,$q^-(p_j)$,$q^+(p_j)$ をそれぞれ,$q(p_i, p_j)$,$q(p_j, p_i)$ と表す. 同様に,$p_i \in \mathbf{MH}$ なら, $MSS\ p_j \in L_i$のとき,$q^-(p_i)$,$q^+(p_i)$ をそれぞれ,$q(p_j, p_i)$,$q(p_i, p_j)$ と表す. 状態 d_iのキュー q_iへメッセージ m を入れた後のキューの状態を $enque(d_i, m)$, 状態 d_iのキュー q_iから先頭のメッセージを取り出した後のキューの状態を $deque(d_i)$ とする. さらに, キュー q_iの先頭にあるメッセージを,$head(d_i)$ と表す. また, 移動管理手続きが交換する移動情報 fについて, fのうち, チャネルの接続関係に関する情報を,$channel\ (f)$ と表す.

- 内部イベント $internal_i(c, c')$
  - (a) ホスト p_iの内部状態のみを Iから I'に変化させる.
  - (b) $L = L'$.
  - (c) $c_i = c$.
  - (d) $c'_i = c'$.
- 送信イベント $send_i(c, m, c')$
  $p_j = dst(m)$,$q_k = q(p_i, p_j)$ とする.
  - (a) ホスト p_iがメッセージ m を送信し, 内部状態を Iから I'に変化させる.
  - (b) $L = L'$,$p_j \in L$.
  - (c) $c_i = c$.
  - (d) $c'_i = c'$,$d'_k = enque(d_k, m)$.
- 受信イベント $receive_i(c, m, c')$
  $p_j = src(m)$,$q_k = q(p_j, p_i)$ とする.
  - (a) ホスト p_iがメッセージ m を受信し, 内部状態を Iから I'に変化させる.
  - (b) $L = L'$,$p_j \in L$.
  - (c) $c_i = c$ かつ $head(d_k) = m$.
  - (d) $c'_i = c'$,$d'_k = deque(d_k)$.
- 切断イベント $disconnect_i(c, p_j, f', c')$
  - (a) L_iを空にし, 内部状態を Iから I'に変化させ, 移動情報 f'を追加する.
  - (b) $p_i \in \mathbf{MH}$,$p_j \in \mathbf{MSS}$,$L = \{p_j\}$,$L' = \emptyset$.
  - (c) $c_i = c$.
  - (d) $c'_i = c'$,$F' = F \cup \{f'\}$
    ここで,$channel\ (f')$ は,p_iの接続状態から p_jが削除されたという情報で,$rmc(p_i, p_j)$ と表す.
- 支援終了イベント $remove_i(c, p_j, f, f', c')$
  - (a) L_iから p_jを削除し, 内部状態を Iから I'に変化させ, 移動情報 fを取得し,f'を追加する.
  - (b) $p_i \in \mathbf{MSS}$,$p_j \in \mathbf{MS}$,$p_j \in L$,$L' = L \backslash \{p_j\}$,$channel\ (f) = rmc(p_j, p_i)$.
  - (c) $c_i = c$ かつ $f \in F$.
  - (d) $c'_i = c'$,$F' = F \backslash \{f\} \cup \{f'\}$
    ここで,$channel\ (f')$ は, どの MSS の接続状態にも p_jが属さないという情報で,$mv(p_j)$ と表す.
- 支援開始イベント $accept_i(c, p_j, f, f', c')$
  - (a) L_iに p_jを追加し, 内部状態を Iから I'に変化させ, 移動情報 fを取得し,f'を追加する.
  - (b) $p_i \in \mathbf{MSS}$,$p_j \in \mathbf{MH}$,$p_j \notin L$,$L' = L \cup \{p_j\}$,$channel\ (f) = mv(p_j)$.
  - (c) $c_i = c$ かつ $f \in F$.
  - (d) $c'_i = c'$,$d'_k = \varepsilon$ (ただし,$q_k = q^-(p_j)$) ,$F' = F \backslash \{f\} \cup \{f'\}$
    ここで,$channel\ (f')$ は,p_iの接続状態に p_jが追加されたという情報で,$mkc(p_i, p_j)$ と表す.
- 接続イベント $connect_i(c, p_j, f, c')$
  - (a) L_iに p_jを追加し, 内部状態を Iから I'に変化させ, 移動情報 fを取得する.
  - (b) $p_i \in \mathbf{MH}$,$p_j \in \mathbf{MSS}$,$L = \emptyset$,$L' = \{p_j\}$, $channel\ (f) = mkc(p_j, p_i)$.
  - (c) $c_i = c$ かつ $f \in F$.
  - (d) $c'_i = c'$,$d'_k = \varepsilon$ (ただし,$q_k = q^+(p_i)$) ,$F' = F \backslash \{f\}$.

□

分散アルゴリズムは, 各ホストに対応する状態機械を定めるものである. これにより, ホストの状態に応じて生起しうるイベントが定義される. しかし, 分散移動システムにおける移動イベントは, ホストの内部状態ではなく, 外的な要因 (例えば, 計算機の物理的な移動など) によって起こると考えられる. 従って, 移動イベントは生起し得るすべての状態に対し用意されなければならない. これは, 例えば,MH p が MSS qを接続状態から削除する切断イベントが, $L = \{q\}$ である任意の状態 c に関して存在することを意味する. 支援終了イベント, 支援開始イベント, 接続イベントについても同様である.

分散移動システムの実行は，状況とイベントの交互列$\gamma^0, e^1, \gamma^1, e^2, \gamma^2, \cdots, e^i, \gamma^i, \cdots$と表される．　ここで，$\gamma^0, \gamma^1, \cdots, \gamma^i, \cdots$ は状況（γ^0は初期状況），$e^1, \cdots, e^i, \cdots$はイベントである．ただし，イベントe^iは状況γ^{i-1}で生起可能なイベントであり，γ^{i-1}でイベントe^iの生起後の状況がγ^iである．以下では，$\mathcal{E}$を任意の実行，Eを$\mathcal{E}$に現れるすべてのイベントの集合とする．ただし，簡単のため，Eに現れるすべてのイベントは相異なるものとする．

定義 2 MH の移動
MH p_iにおいて，ある切断イベントが起きた後，次の接続イベントが起こるまでの間，すなわち，接続状態$L_i = \emptyset$のとき，*MH* p_iは移動中であるという．　□

任意の実行に対し，以下の公正さを仮定する．

仮定 1 イベント生起の公正さ
実行$\mathcal{E}$中の連続する無限個の状況に対し，すべての状況であるメッセージmを送信するイベントが生起可能であれば，そのうちのいずれかの状況が生起している．あるホストのある内部状態で生起可能な内部イベント，あるメッセージを受信する受信イベント，ある*MH*を接続状態から削除する支援終了イベント，ある*MH*を接続状態に追加する支援開始イベント，ある*MSS*を接続状態にする接続イベントに関しても，同様に仮定する．　□

MH の移動は，4 つの移動イベント (*disconnect*, *remove*, *accept*, *connect*) でモデル化される．これら 4 つのイベントを ME と書く．$\mathcal{E}$に現れるすべての ME の集合を **ME** とする．

次に，Lamport の前後関係 (causal order)[8] を拡張し，分散移動システムの実行に現れるイベント間の前後関係を定義する．ここで，すべてのメッセージ，すべての移動情報は相異なると仮定し，メッセージmの送信イベントを$send(m)$，受信イベントを$receive(m)$と表す．特に，ホストp_iのイベントであることを表すときは，$send_i(m)$，$receive_i(m)$と表す．

定義 3 前後関係
E上の前後関係$\prec$は，以下を満たす最小の 2 項関係である．

(1) イベントeとe'が同じホストの異なるイベントで，$\mathcal{E}$においてeがe'より前に現れるとき，$e \prec e'$である．

(2) メッセージmについて，$send(m)$と$receive(m)$が共にEに属するなら，$send(m) \prec receive(m)$である．

(3) 任意の (*disconnect*, *remove*, *accept*, *connect*) $\in$ **ME** について，
$disconnect \prec remove \prec accept \prec connect$ である．

(4) 推移律が成り立つ．すなわち，
$(e \prec e') \wedge (e' \prec e'') \Longrightarrow e \prec e''$である．

□

MH を含まない通常の分散システムと同様に [12]，無矛盾な切断，無矛盾な状況を次のように定義する．

定義 4 無矛盾な切断
Eの部分集合Cが，$\mathcal{E}$の無矛盾な切断であるとは，任意の$e, e' \in E$に対し，

$$(e \in C) \wedge (e' \prec e) \Longrightarrow e' \in C$$

が成り立つことである．　□

Mをある 1 つのホストが送信したメッセージの集合とする．Mの要素を送信順に並べた系列を$seq(M)$と表す．

定義 5 無矛盾な状況
状況$\gamma = (c_1, \ldots, c_N, d_1, \ldots, d_M, F)$に対し，以下の条件を満たす$\mathcal{E}$の無矛盾な切断$C$が存在するとき，$\gamma$は実行$\mathcal{E}$の無矛盾な状況である．

(1) 各ホストp_iに対し，f_iをCの中のp_iのイベントで最後に生起したイベントであるとすると，f_iが存在するとき，f_iはp_iの状態をc_iに変化させ，f_iが存在しないとき，c_iはp_iの初期状態である．

(2) 静的チャネルのキュー$q_i = q(p_j, p_k)$に対し，$d_i = seq(\{m | send_j(m) \in C \wedge receive_k(m) \notin C\})$．

(3) 動的チャネルのキュー$q_i = q^-(p_j)$ $(= q(p_k, p_j))$とする．　ここで，*MSS* p_kが存在しない場合，p_kを *MH* p_jの最後に隣接していた *MSS* とする．Cの中で *MSS* p_kで最後に生起した *MH* p_jの支援開始イベントを，$last_accept_k$とすると，$d_i = seq(\{m | last\_accept_k \prec send_k(m) \wedge send_k(m) \in C \wedge receive_j(m) \notin C\})$．

(4) 動的チャネルのキュー$q_i = q^+(p_j)$ $(= q(p_j, p_k))$とする．　ここで，*MSS* p_kが存在しない場合，p_kを *MH* p_jの最後に隣接していた *MSS* とする．Cの中で *MH* p_jで最後に生起した *MSS* p_kとの接続イベントを，$last_connect_j$とすると，$d_i = seq(\{m | last\_connect_j \prec send_j(m) \wedge send_j(m) \in C \wedge receive_k(m) \notin C\})$．

(5) 移動情報fを追加するイベントを，$put(f)$，fを取得するイベントを$get(f)$と表す．　このとき，$F = \{f | put(f) \in C \wedge get(f) \notin C\}$．

□

分散移動システムでは，接続状態の変化は，チャネルの両端のホストで非同期に行なわれる．すなわち，ある瞬間には動的チャネルの一端だけが存在する状況が生ずる．このような状況を形状的に矛盾する状況と考え，無矛盾かつ形状的無矛盾な強無矛盾状況を定義する．

定義 6 強無矛盾な状況
状況$\gamma = (c_1, \ldots, c_N, q_1, \ldots, q_M, F)$が以下の条件を満たすとき，$\gamma$を実行$\mathcal{E}$の強無矛盾な状況という．

- γは$\mathcal{E}$に対し無矛盾．
- ホストp_iがホストp_jの接続状態に含まれるなら，ホストp_jはホストp_iの接続状態に含まれる．すなわち，$\forall i, j\ [p_i \in L_j \Longleftrightarrow p_j \in L_i]$．

□

3 スナップショットアルゴリズム

3.1 定義

スナップショットアルゴリズムは, 分散移動システムにおいて, 強無矛盾な状況を求めるための分散アルゴリズムである. このアルゴリズムは, 状態機械 $k_1, \ldots, k_N$ からなるとする. 任意の分散移動システム Sに対し,Sの各ホスト p_iを, p_iと k_iからなるホスト p_i'に置き換えた分散移動システム S'を考える. p_i'の状態は,p_iと k_iの状態の対で,p_iまたは k_iのイベントにより状態が変化する. S'の実行 $\mathcal{E}'$は,Sの実行 $\mathcal{E}$に並行して, 各 k_iの状態変化や k_i間でのメッセージや移動情報の交換が行なわれているとみなせる. 各 k_iは p_iの状態や p_iに接続するチャネルの状態を記録するイベントをもち, $\mathcal{E}'$において記録されたすべての状態は, $\mathcal{E}$の強無矛盾な状況を形成する. スナップショットアルゴリズムで求められた強無矛盾な状況をスナップショットと呼ぶ.

本稿では, 任意の1つのホストがスナップショットアルゴリズムを起動すると仮定する. このホストを起動ホストと呼ぶ. また,MH は移動中にはスナップショットアルゴリズムを起動することはないと仮定する. MH が移動中の場合, この MH は MSS ネットワークに接続していない. このような状況で MH がシステム全体の状況を必要とすることはないと考えられるので, この仮定は現実的である.

任意の実行 $\mathcal{E}$の初期状況γ^0は $\mathcal{E}$の強無矛盾な状況である. 従って, スナップショットとして初期状況γ^0を求める自明なスナップショットアルゴリズムも考えられる. これを避けるために, 求めるスナップショットは起動ホストがスナップショットアルゴリズムを起動したときの状態 (以下, 起動状態と呼ぶ) を含むものとする.

定理 1 スナップショットの存在性
任意の分散移動システムの, 任意の実行を $\mathcal{E}$とする. $\mathcal{E}$の任意の状況における任意のホスト pの状態 cについて, その状態を含む強無矛盾な状況が存在する. □

3.2 分散移動システムにおける仮定

Chandy と Lamport は MH を含まない分散システム上のスナップショットアルゴリズム (以下,*CL* アルゴリズムとする) [4] を提案している. 本稿では,*CL* アルゴリズムを分散移動システム上でも動作するように拡張したスナップショットアルゴリズム (以下,$\mathcal{MS}$アルゴリズムとする) を提案する.

CL アルゴリズムでは,*marker* と呼ばれるメッセージを用いて, 各ホストが相互に協調しながらスナップショットを求める. 各ホストは, 自発的に, または,*marker* を受信したときにアルゴリズムを開始する. アルゴリズムを開始したホストは, まずその状態を記録し, 次にすべての隣接ホストへ *marker* を送信する. 伝送中のメッセージは, 送信ホストが状態を記録する前に送信され, 受信ホストが状態を記録した後に受信されるメッセージである. 従って, ホストが状態を記録した後に受信するメッセージのうち, *marker* より前に受信したものであり, 受信ホストがこれらを記録する.

分散移動システムのスナップショットアルゴリズムでは, 強無矛盾な状況を記録しなければならない. また, 動的チャネルを伝送中のメッセージの消失に対処しなければならない. このために,*CL* アルゴリズムでは必要としない, いくつかの仮定を設ける必要がある.

例えば, MSS pが, 状態 $c = (I, L)$でアルゴリズムを起動し, 状態 cを記録したとする. $p' \in L$ である MH p'で既に切断イベントが生起している場合を考える. p'が記録する状態を $c' = (I', L')$ とすると, スナップショットが強無矛盾であるためには,$p \in L'$でなければならない. しかし, これは,p がアルゴリズムを開始する前の状態である. p がアルゴリズムの起動を知るのは, 早くとも, 次の接続イベント時である.

また,*CL* アルゴリズムでは, チャネルを伝送中のメッセージの記録は受信側で行なっている. これは, 送信されたメッセージは必ず受信されるという性質に基づいている. しかし, 分散移動システムの動的チャネルに送信されたメッセージは, その接続関係の変化のために消失することがあり, その受信は保証されない (これは, 支援開始イベント, 接続イベントで,MH に接続するチャネルの内容を空系列にしていることに対応する). 従って, 伝送中のメッセージの記録の一部は, 送信側で行なう必要がある. しかし, これらは, 状態を記録する前に送信したメッセージであるので, 起動ホストにとっては, アルゴリズムの開始する前に送信したメッセージであるので, アルゴリズムが記録することができない.

そこで, 本稿では, 以下の仮定を設ける.

- 移動中の MH pは, 最も最近の切断イベントについて, その直前の状態を知っている.
- MSS pで支援終了イベント $remove(c, p', f, f', c')$ が起こるとき, MSS pから MH p'へ送信したメッセージのうち, MH p'が受信しなかったメッセージを MSS p は知っている.
- MH pで接続イベント $connect(c, p', f, c')$ が起こるとき, MH pから pの移動前の隣接 MSS (p''とする) へ送信したメッセージのうち, p''が受信しなかったメッセージを MH p は知っている.

3.3 $\mathcal{MS}$アルゴリズムの概要

$\mathcal{MS}$アルゴリズムでは, 各ホストは, 自発的に, または,*marker* の受信, あるいは, 移動情報の取得によって, アルゴリズムを開始する. アルゴリズムを開始したホストは, そのときの状態を記録するか, または, 強無矛盾性を満たすために, 保存している過去の状態を記録する. 過去の状態を記録するのは, 切断イベントの生起時に MH において状態を記録しておらず, MSS において対応する支援終了イベントが生起する前に, その MSS が状態を記録している場合である. このとき, 接続イベントが生起するときに取得する移動情報によって, MH は, 移動前の MSS が支援終了イベントの起こる前の状態を記録したことを知り, 切断イベントの直前の状態を記録する.

次に, 伝送中のメッセージの記録について述べる. ホスト p_i,p_jが記録した状態を c_i,c_jとする. c_i,c_jにおいて,p_iと p_jの間にチャネルが存在しているならば, そのチャネルについて伝送中のメッセージを記録する必要がある. p_iから p_jへ伝送中のメッセージは,c_iで送信済みで,c_jで未受信のものである. *CL* アルゴリズム同様,$\mathcal{MS}$アルゴリズムでも, 状態を記録した後,*marker* を送信する. あるチャネルから

$marker$を受信する場合, そのチャネルを伝送中のメッセージは, CLアルゴリズム同様, すべて受信ホストが記録する. p_jが記録した状態$c_j = (I_j, L_j)$において, $p_i \in L_j$であるが,p_jがp_iから$marker$を受信しない場合について考える.

(1) $p_j \in$ **MH** の場合 : p_jが$marker$を受信する前にp_jで切断イベントが生起したために,p_jはp_iからの$marker$を受信しない.

  (a) MH p_jが切断イベントの生起時に既に状態を記録している場合 : MH p_jが状態を記録してから, 切断イベントが生起するまでに受信したメッセージは, p_iからp_jに伝送中のメッセージとして記録できる. 切断イベントが生起するまでに,p_jが受信できない伝送中のメッセージは, 仮定により,p_iが記録できる.

  (b) MH p_jが切断イベントの生起時に状態を記録していない場合 : p_jが記録した状態c_jにおいて,p_iが隣接MSSなので, この場合,p_jはアルゴリズムの開始時に切断イベントの直前の状態を記録する. p_iからp_jへ送信された伝送中のメッセージは,p_jが状態を記録した以降に受信するメッセージである. これらはすべて,p_jでは受信できないが, 仮定により, p_iで支援終了イベントが起こるときに記録できる.

(2) $p_j \in$ **MSS** の場合 : $marker$を受信しないことから,p_jにおいて, MH p_iの支援終了イベントが生起する. $p_i \in L_j$なので, この支援終了イベント生起時には,p_jは状態を記録している. このとき,MSS p_jが状態を記録してから, 支援終了イベントが生起するまでに受信したメッセージは,p_iからp_jに伝送中のメッセージとして記録できる. 支援終了イベントまでに, p_jが受信できない伝送中のメッセージは, 仮定により,MH p_iで接続イベントが起こるときに記録できる.

図2, 図3に,$\mathcal{MS}$アルゴリズムを$Pascal$風のプログラムで示す. $\mathcal{MS}$アルゴリズムでは, 以下の変数, メッセージ, 移動情報を用いる.

各ホスト p の変数

- $taken_p$: 論理型変数. 初期値は$false$で, ホスト p が状態を記録した場合に$true$である.
- $tkch_p[p']$: 各 $p' \in L_p$について用意される論理型変数. 初期値は$false$. $p \in$ **MH** の場合は,L_pの要素は高々1つなので,$tkch_p$と表す. ホスト p'からホスト p へ送信された伝送中のメッセージのうち, p が記録すべきものについて記録した場合に$true$である.

メッセージ

- $marker$: CLアルゴリズムと同様に用いる.

移動情報

$\mathcal{MS}$アルゴリズムは移動情報を使用する. 移動情報 fのうち,$\mathcal{MS}$アルゴリズムが使用する情報を,$MS(f)$と書く. $MS(f)$は, 次の2種類存在する.

- $old(p)$: MH p が切断イベントを起こす前に状態を記録したこと, もしくは,MH p が切断イベント直前の状態を記録しなければならないことを表す情報.
- $iso(p)$: MH p は移動中の状態を記録しなければならないことを表す情報.

スナップショットアルゴリズムの開始

ホストは次のいずれかによって, アルゴリズムの実行を開始する.

(1) 自発的にアルゴリズムを起動する（起動ホストの場合）.

(2) ⟨**mkr**⟩ を受信する.

(3) 支援終了イベントの生起で取得する移動情報 fについて,$MS(f) = old(p')$である（MSSのみ）.

(4) 支援開始イベントの生起で取得する移動情報 fについて,$MS(f) = old(p')$である（MSSのみ）.

(5) 接続イベントの生起で取得する移動情報 fについて,$MS(f) = old(p)$, あるいは, $MS(f) = iso(p)$である（MHのみ）.

アルゴリズム

各MSS pが実行するアルゴリズムを図2, 各MH pが実行するアルゴリズムを図3に示す. 図2,3において, $wait(event)$は生起可能な p のイベントの中から任意の1つを選び, そのイベントに関する情報を引数$event$として返す手続きである. ただし, 生起可能な p のイベントが存在しなければ, 1つ以上のイベントが生起可能になるまで待つ. また, ホスト p が起動ホストでなく,p がアルゴリズムの実行を,$marker$受信, 支援終了イベント, 支援開始イベント, 接続イベントのいずれかによって開始する場合, 最初の$wait(event)$で, その最初のイベントに関する情報が引数$event$に返されるものとする. 引数eventに返される情報は, 以下のいずれかである.

(a) $receive_{msg}(p', m)$:
$p' \in L_p$からのメッセージ m の受信イベント.

(b) $receive_{mkr}(p', m)$:
$p' \in L_p$から$marker$の受信イベント.

(c) $disconnect(p')$:
MSS $p' \in L_p$に関する切断イベント.

(d) $remove(p', f)$:
MH $p' \in L_p$に関する支援終了イベント.

(e) $accept(p', f)$:
MH $p' \notin L_p$に関する支援開始イベント.

(f) $connect(p', f)$: MSS $p' \notin L_p$に関する接続イベント.

ただし,(d),(e),(f) において,fはこのイベントの生起で取得する移動情報を表す. また f'は, そのイベントの生起により追加される移動情報を表す.

4 あとがき

分散アルゴリズムの設計のための分散移動システムのモデルを提案し, スナップショット問題という基本的, かつ, 重要な問題について, 分散移動システム上のアルゴリズム

```
var   taken_p      : boolean init false ;
      tkch_p[p']   for each p' ∈ L_p : boolean init false ;

begin
  record the current state c*_p = (I*_p, L*_p) ;
  taken_p := true ;
  for all p' ∈ L*_p do
    send marker to p' ;
  repeat forever do
  begin
    wait(event) ;
    case event of
      receive_msg(p', m) :
        if tkch_p[p'] = false then record m ;
      receive_mkr(p') :
        tkch_p[p'] := true ;
      remove(p', f) :
        begin
          if p' ∈ L*_p then
          begin
            MS(f') := old(p') ;
            record the messages sent to p' before marker
                                    but not received by p' ;
            tkch_p[p'] := true ;
          end
        end ;
      accept(p', f) :
        if MS(f) = old(p') then MS(f') := old(p') ;
        else /* MS(f) = null */ MS(f') := iso(p') ;
    end
  end
end.
```

図 2: The snapshot algorithm for MSS p

```
var   taken_p      : boolean init false ;
      tkch_p       : boolean init false ;

begin
  if p is initialized by a connect event
                              with MS(f) = old(p) then
    record the stored state (i.e., the state immediately
                              before the last disconnect event) ;
  else record the current state ;
  taken_p := true ;
  if p is initialized spontaneously or by receipt of marker then
    send marker to the local MSS ;
  repeat forever do
  begin
    wait(event) ;
    case event of
      receive_msg(p', m) :
        if tkch_p = false then record m ;
      receive_mkr(p') :
        tkch_p := true ;
      connect(p', f) :
        begin
          if MS(f) = old(p) then
            record messages sent to the previous local MSS p''
                              before marker but not received by p' ;
          tkch_p := true ;
        end ;
      disconnect(p') :
          MS(f') = old(p) ;
    end
  end
end.
```

図 3: The snapshot algorithm for MH p

を提案した. 提案したモデルでは, 計算機の移動を動的チャネルの両端で非同期に起こる接続関係の変化で表している. よって, ある状況では, 分散移動システムの各ホストの隣接関係の認識に矛盾が生じる場合がある. 本稿では, 従来のメッセージの送受信関係に基づく無矛盾性に, この接続関係の無矛盾性を加えた状況の強無矛盾性を定義した. また, 提案したスナップショット問題を解くアルゴリズムでは, この強無矛盾性を考慮している. 紙面の都合で省略したが, 我々は, 提案したアルゴリズムの正当性の証明, および計算複雑度の解析も行なった.

謝辞：日頃より, 分散アルゴリズムについて熱心にご討論いただく本学情報科学センターの片山喜章助手, 本学藤原研究室の藤原暁宏氏, 松井博義氏に感謝致します. また, 井上智生助手を始めとする本学藤原研究室の諸氏に感謝致します. なお, 本研究の一部は, 文部省科学研究費補助金（奨励 (A)07780271）, 財団法人電気通信普及財団の助成による.

参考文献

[1] A. Acharya and B. R. Badrinath. Delivering multicast messages in networks with mobile hosts. *Proc. 13th ICDCS* (1993).

[2] A. Acharya and B. R. Badrinath. Checkpointing distributed applications on mobile computation. *Proc. 3rd International Conf. on Parallel and Distributed Infomation Systems*, pp.73-80 (1994).

[3] 青柳, 真鍋. 分散デバッガのためのチェックポイント・ロールバックアルゴリズム, 日本ソフトウェア科学会ソフトウェア研究会（関西）資料, SW-92-10-4 (1992).

[4] K.M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Trans. Comput. Systems*, 3,1,pp.63-75 (1985).

[5] J. Ioannidis, D. Duchamp, and G. Q. Maguire. Ip-based protocols for mobile internetworking. *Proc. ACM SIGCOMM Sympo. on Communication. Architechtures and Protocols*, pp.235-245 (1991).

[6] D. B. Johnson and W. Zwaenopoel. Recovery in distributed systems using optimistic message logging and checkpointing. *Proc. 7th PODC* (1988).

[7] 亀田, 山下. 分散アルゴリズム, 近代科学社 (1994).

[8] L. Lamport. Time, clocks and ordering of events in a distributed system. *CACM*, 21, 7, pp.558-564 (1978).

[9] T. H. Lay and T. H. Yang. On distributed snapshot. it IPL 25,pp.153-158 (1987).

[10] 増澤, 都倉. 分散デバッガのための Causal Distributed Breakpoint を求めるアルゴリズム, 1992年電子情報通信学会秋季大会, SD-1-8 (1992).

[11] R. E. Strom and S. Yemini. Optimistic recovery in distributed systems. *ACM Trans. on Computer Systems*,3,3 (1985).

[12] G.Tel. Intoroduction to Distributed Algorithms. *Cambridge University Press* (1994).