近傍オブジェクトモデル

地引　昌弘[†a)]　　久野　靖[†b)]

Neighborhood Object Model

Masahiro JIBIKI[†a)] and Yasushi KUNO[†b)]

あらまし　分散プログラミングをミドルウェアによりサポートする既存の技術では，ノードの配置が変化するとプログラムの変更も生ずるなど，ノードとノードの境界を強く意識したプログラミングが必要となる．一方，上位層にノードの境界を意識させない分散OSは，強力な透過性を提供する有力な方式であるものの，稼働環境が均質でない場合や上位層で実行位置を意識する必要がある場合などに弱点も生じる．つまり，非均質プラットホーム/非一様ネットワークにおけるプログラミングでは，論理的な透過性だけでなく，必要があれば物理的環境を意識できるプログラミングパラダイムも重要となる．本論文では，実行環境の不均質性を近傍という形でモデル化して扱い，以下のアイデアにより透過性の提供と実行環境の最適化を両立する方式を提案する．(1) 導入の容易な仮想マシン (VM：Virtual Machine) を各ノード上に配置し，全体で共通の分散実行環境を提供する．(2) 各分散VMは，物理的な実行環境における近傍を認識する．(3) オブジェクトへのアクセスが生じると，分散VM間で近傍を意識した最適なオブジェクトを動的に探索する．更に本論文では，提案方式をインターネットのような非均質環境に適用した場合の有効性についてシミュレーションによる評価を行う．

キーワード　インターネット，分散環境，オブジェクト，透過性

1. まえがき

インターネットに代表されるネットワーク技術の発達に伴い，複数のノードを用いてシステムを構築する分散システムが広く用いられている．これらの例としては，複数のクライアントがサーバにサービスを要求するクライアントサーバ型システムや，複数のサーバ上をエージェントが順次移動しつつ処理を行う移動エージェント型システムなどがあげられる．特に最近では，インターネットにおける利用を意識したシステムの構築をサポートするミドルウェアとして，CORBA [6], [13] や Java RMI [8], [9] に代表される分散オブジェクト技術 [14], [18] が普及している．しかし，これらの技術では，現在いずれもノードとノードの境界を強く意識したプログラミングを行わざるをえない．これにより，環境内でオブジェクトを共有し合うような分散システムでは，ノードの配置に変化が生じるとプログラムも変更する必要があるなど，分散化が不利に働くことも予想される．これに対し，ノードの境界を意識せずにプログラミングを行える環境として分散OSがあげられる．しかし，分散OSは強力な透過性を提供するものの，OSの置換えは現実には困難といえる．

更に，ノードの境界を完全に見えなくしてしまうことは，次のような問題も生じさせる．

- 稼働環境が均質でない場合に，高速なノードやネットワークの利用をユーザあるいはプログラム側から指定できない．
- セキュリティ上の理由などにより上位層で実行位置を意識する必要があっても，これをユーザあるいはプログラム側から検出できない．

すなわち，インターネット環境のような非均質プラットホーム/非一様ネットワークにおけるプログラミングでは，論理的な透過性だけでなく，必要があれば物理的環境を意識できるプログラミングパラダイムも重要である．

本論文では，実行環境の不均質性を近傍という形でモデル化して扱い，以下のアイデアにより透過性の提供と実行環境の最適化を両立する方式を提案する．

† 筑波大学，東京都
Tsukuba University, 3–29–1 Otsuka, Bunkyo-ku, Tokyo, 112–0012 Japan
a) E-mail: jibiki@bx.jp.nec.com
b) E-mail: kuno@gssm.otsuka.tsukuba.ac.jp

- 導入の容易な仮想マシン (VM：Virtual Machine) を各ノード上に配置し，全体で共通の分散実行環境を提供する．
- 各分散 VM は，物理的な実行環境における近傍を認識する．
- オブジェクトへのアクセスが生じると，分散 VM 間で近傍を意識した最適なオブジェクトを動的に探索する．

以下，**2.** では分散環境における透過性の問題点について検討する．続く **3.** では，**2.** で述べた問題を解消する近傍オブジェクトモデルの提案を行う．その後，**4.** で近傍オブジェクトモデルに関する評価を行い，最後に **5.** で議論とまとめを行う．

2. 分散環境と透過性

2.1 分散環境におけるプログラミング

分散環境における現在のシステム構築では，ノードとノードの境界を強く意識したプログラミングが必要となる．例えば，クライアントサーバ型システムでは遠隔手続きの呼出し (RPC) が明示的にノードの境界を表し，そこで渡される引数や返値の扱いは他の部分における扱いと明確に異なっている．レコード型などの複雑なデータ構造を渡す場合，通常の関数呼出しではアドレス渡しを行えるが，RPC ではポインタを渡すことができないため，すべて値渡しに変更する必要がある．移動エージェントについても，ノード間におけるエージェントの移動は明示的に指示され，移動後も特定位置から実行再開される形式が一般的である [15]．例えば，“go (行き先)” といった命令で移動が発生しても，移動後はその命令の次から実行を再開するわけではなく，別に定義されたプログラムの先頭から再開する．再開したプログラム上では移動に関する状況をチェックし，実際の処理を再開する場所を決める．

一方，CORBA や Java RMI に代表される分散オブジェクト技術は，システムの各要素をオブジェクトとして統一的に扱うことにより，オブジェクト指向の枠組みを用いて抽象度を高め，より高い透過性を提供している．しかし，RMI では，基本データ型や直列化可能なオブジェクトが値渡しされるのに対し，リモートオブジェクトは参照渡しされ，これら以外のオブジェクトは渡すことができないなど，ノードの境界が完全にいんぺいされているわけではない．更に，RMI を受け入れるリモートオブジェクトの参照を取得するには，特定のオブジェクトを名指しする情報（オブジェクトの固有名）を用いて，次のようにユーザがオブジェクトの位置を決定しなければならない．

- URL などを用いて直接指定する (Java RMI, HORB [3], [4])．
- 一意的な識別子などをキーにして名前サーバを引く (CORBA)．
- 大域的な名前を定義する [2]．

これは，ユーザがオブジェクトを利用する場合に，前もって対応するオブジェクトの固有名も取得しておく必要があり，自分に必要なサービスを把握しているだけでは適切なサービスを受けられないことを意味している．これらのサービスには，例えば DNS サーバにより提供される名前解決や，ミラーリングサーバを用いたファイルのダウンロード，メールサーバにより行われるメール転送など，効率化をめざして同様な機能を提供する複数のサーバを環境内の広範囲に配置するサービスが主に該当する．

以上より，分散環境におけるプログラミングでは，その繁雑さを減らすために次のことが望まれる．

（1） 必要なサービスを指定するだけで該当するオブジェクトを取得できる．

（2） ノードの境界を意識することなく他のオブジェクトと通信できる．

これらを実現するには，言語の機能として以下のいんぺい機構を提供する必要がある [10], [11]．

（i） 通信処理のいんぺい

リモートオブジェクトへの通信をローカルな通信に見せることで，リモートオブジェクトとローカルオブジェクトを区別せずに扱える．

（ii） 固有名のいんぺい

システムがオブジェクトの場所を管理し，指定されたサービスを提供できるオブジェクトを自動的に参照する．

2.2 透過性の限界

分散オブジェクト技術とは別に，より強力な透過性を提供する方式として，1980 年代に多く研究された分散 OS の考え方がある．分散 OS では，複数のノードを合わせたものが一つのシステムとして稼働するため，そのうえで動作するアプリケーションはノードの境界を意識する必要がなく，また，分散 OS からサービスを受けるプロセスは，ノード間を移動してもそれを関知しない．分散 OS が提供する透過性は，プログラミングの複雑さを大きく減らすことに成功したが，現状の OS をすべて分散 OS へ置き換えることは現実

的でないことから，普及することはなかった．

また，分散OSのような完全な透過性の提供は，その代償として以下のような弱点も生じさせる．

(a) 稼働環境が均質でない場合の問題点

稼働環境が均質でない場合は，一般に透過性を高めると実行効率のばらつきが大きくなる．例えば，インターネットのような大規模開放型環境では，ネットワーク状況やサービスの分布状況など物理的な条件に開きが大きく，その組合せによっては実行効率に大きな差が生じる．更に，プログラミングに際しても，規模が大きく一様でないネットワークでは遅延が生じやすく，返信が来ない場合はエラーによるものか遅延によるものかを判断することが難しい．その結果，不確実な処理を排除するには，通信のタイムアウトを定義してタイムアウトが発生した場合の処理を追加するなど，プログラミングにも特別な工夫が必要となる．

(b) 実行位置を意識する必要がある場合の問題点

透過性が価値をもつのは，計算資源の存在位置やその処理位置を問わずにサービスを提供できる場合である．しかし，すべての計算資源やその処理が位置を問わずにサービスを提供できるわけではなく，種類によってはその実行位置を特定する必要がある．例えば，セキュリティ上，いんぺいすべきデータを操作する場合は，そのデータの保管場所で操作すべきである．物理的に離れた場所で操作を行う場合は，いんぺいすべきデータがネットワークを流れることになり，盗聴などによるセキュリティの低下や暗号化などのオーバヘッドが必要になる．同様に，データサイズが大きい場合なども，データを移動させるよりそのデータの保管場所で操作したほうが全体として効率が良く，実行位置を意識したプログラミングが必要となる．

以上のように，環境内の計算資源に対する完全な透過性が常に望ましいというわけではなく，周囲の物理的状況や計算資源及びその処理の種類に応じて，制限する必要もある．

3. 近傍オブジェクトモデル

前章末で述べたように，インターネットのような非均質環境におけるプログラミングでは，透過性の提供に加え，周囲の物理的環境を意識できるプログラミングパラダイムに基づいた新しいプログラミングモデルの構築が必要となる．また，そのような周囲の物理的環境を意識したオブジェクトモデルは，OSの置換えなど大がかりな準備を必要とせず，容易に実現可能な方式として提供することが望まれる．本論文では，プログラミング言語あるいはプログラミングモデルに対して，近傍の概念を導入することにより，透過性の提供と実行環境の最適化を両立させる方式を提案する．

3.1 オブジェクトと近傍

インターネットに代表される大規模開放型[18]の分散環境で効率的なシステムを構築するには，実行環境に対する最適化を行う必要がある．実行環境の最適化は，オブジェクトの実装を選択する際に，通常のインタフェース情報だけではなく実行環境の物理的条件も利用することで実現できる．

イントラネットのような小規模で閉鎖的/静的な環境では，環境内の位置の違いにより物理的条件に大きな差がある場合は少ない．したがって，負荷の偏りによる実行効率の低下と最適化に要するオーバヘッドとを比較すると，通常は後者のほうがオーバヘッドは高くなる．このような場合にオブジェクトを近接させると，近接による通信時間の短縮よりもオブジェクトの移動にかかる時間のほうが長くなり，オブジェクトアクセスに対する総所要時間は大きくなる．これに対し，インターネットに代表される非均質環境では，環境内の物理的条件の差が極めて大きく，そのため最適化の余地は多く残されている．例えば，インターネット上の通信は条件に応じて遅延が非常に大きくなることから，イントラネットに比べてオブジェクトの近接による通信時間の短縮効果は大きくなる．

以上の議論より，オブジェクトには実行効率に応じた距離の概念を導入できる．しかし，実行効率には様々な要因が関連するため，一意な値として距離を定義することは問題も多く，周辺の領域という意味で近傍を対象にしたほうが実用的といえる．本論文では，オブジェクトに対する近傍を以下のように定義する．

（i） オブジェクトの近傍

ローカルノードを N_l，リモートノードを N_r とした場合，分散環境においてオブジェクト O の実行効率を規定する以下の各パラメータを考える．

$PC_r(O)$： N_r 上で O を実行する場合の実行コスト

$NC_{lr}(O)$：N_l〜N_r 間の通信コスト

$PC_l(O)$： N_l 上で O を実行する場合の実行コスト

$NC_{ll}(O)$：N_l から N_l 上の O に対する通信コスト

$MC_{lr}(O)$：O を N_r から N_l へ移動させる通信コスト

これらが次の式を満たすならば，オブジェクト O はノード N_l の近傍内に存在する．

$$PC_r(O) + NC_{lr}(O) \leqq PC_l(O) + NC_{ll}(O) + MC_{lr}(O) \quad (1)$$

ユーザは，近傍に存在するオブジェクトを利用する，あるいは近傍外のオブジェクトを近接させることにより，オブジェクトの実行効率を上げられる．本論文では以後，オブジェクトの近傍を意識したプログラミングモデルを近傍オブジェクトモデルと呼ぶ．

（ii） 近傍オブジェクトモデル

近傍に基づいた効率化を行う分散オブジェクト環境のプログラミングモデル．

従来の分散オブジェクトモデルでは，異なるノードに分かれて存在するオブジェクトどうしの相互通信に時間が必要なことは考慮するものの，所要時間の程度までは考慮していない．近傍オブジェクトモデルは，分散オブジェクトモデルに対して，オブジェクト間の近さ（通信コストの大小）を考慮するように拡張したモデルといえる．

近傍オブジェクトモデルの実現には，

- 環境内の各ノードが近傍を判断するための情報を取得できること．
- 近傍の内外に応じたオブジェクトを選択/移動できること．

が必要となる．これらは，以下のアイデアにより実現する．

- 導入の容易な仮想マシン (VM：Virtual Machine) を各ノード上に配置し，全体で共通の実行環境を提供する．
- 各分散 VM は，物理的な実行環境における近傍を認識する．
- オブジェクトへのアクセスが生じると，分散 VM 間で近傍を意識しつつ最適なオブジェクトを動的に探索する．

近傍オブジェクトモデルでは，分散 VM による実行環境の共有とオブジェクトアクセス時の動的探索が重要となる．

一方，式 (1) に基づく近傍の判断を実装する際には注意が必要である．オブジェクトの利用者は，オブジェクトの実行にあたり物理的条件を参照する手段をもたないことが多く，近傍の判断を利用者に委ねることは，逆に利用者の負担を増やすおそれもある．したがって，実際の実行環境に関する情報は実行系（例えば分散 VM）で管理し，ユーザは希望する物理的条件を実行系に示すことにより，最適なオブジェクトへのアクセスを獲得できることが望ましい．

3.2 オブジェクトの動的探索

近傍オブジェクトモデルでは，オブジェクトの実行効率から判断される近傍内のオブジェクトを利用することで，実行効率の向上をめざす．しかし，オブジェクトの近傍はその実行時刻や周囲の状況により異なるため，実際にオブジェクトへのアクセスが発生した段階で，アクセス先のオブジェクトが近傍内に存在するかどうかを判断する必要がある．この判断は，各ノード上に配置された分散 VM が，オブジェクトアクセスの発生した時点で，仮想空間内の近傍及び最も近いアクセス先を決定することにより実現できる．

これらの決定には，分散 VM 間で実行環境の情報を交換する必要がある．このような情報の交換には以下の 2 方式が考えられる．

(a) 定期的あるいは実行環境に変化が生じた時点で，情報を交換する．

(b) オブジェクトへのアクセスが発生した時点で，情報を交換する．

最も近いオブジェクトを決定する速度は，オブジェクトアクセスが発生する前に該当するオブジェクトの位置をあらかじめ取得しておくことにより，向上させられる（方式 (a)）．

例えば，一定時間ごとあるいはイベント発生を契機にオブジェクトの位置情報を分散 VM 全体で交換し，各 VM はそれらをキャッシュしておくことで，必要なオブジェクトの位置を瞬時に決定できる．この方式は，経路制御プロトコルの導入によるネットワーク管理と類似しており，オブジェクト位置の変化が各分散 VM のキャッシュへ反映されるまでの時間差及びオブジェクト位置の交換に費やす通信量が問題となる．全世界の網羅的な情報（経路情報）を個々の構成ノードが分散管理するインターネットでは，あるノードで管理する経路情報が変更された場合に全世界のノードへ反映されるまでの時間差は現状数分以下である．しかし，オブジェクトの位置情報を常時交換する方式では，経路制御プロトコルの実装と同様に分散 VM の実装が複雑化し，また，各 VM 間のトポロジーや情報の流れを管理する必要があるなど，運用コストも高くなる．これに対し，分散 VM の構築する仮想空間がより小規模な場合では，アクセスが発生した時点であらかじめ決めておいた場所を探索する方式（オンデマンド探索）も有効である（方式 (b)）．

近傍の決定は **3.1** の式 (1) により行うが，各パラ

メータ (PC_r, NC_{lr}, PC_l, NC_{ll}, MC_{lr}) の決定には以下の情報が必要となる．

他ノードの負荷：PC_r の決定に必要
自ノードの負荷：PC_l の決定に必要
オブジェクトサイズ：MC_{lr} の決定に必要
通信遅延：NC_{lr} 及び MC_{lr} の決定に必要
有効帯域：NC_{lr} 及び MC_{lr} の決定に必要

近接させたオブジェクトに対する通信遅延は，ほぼ 0 とみなすことができるため，自ノードの負荷が極めて大きくない限り $NC_{ll} = 0$ と考えることができる．

4. 評　　価

本章は，近傍オブジェクトモデルの有効性について評価を行う．まず **4.1** では，複数の分散 VM /オブジェクト/利用者が混在する環境において（以下，この環境を系と呼ぶ），近傍オブジェクトモデルを適用した場合と適用しない場合とのオブジェクトアクセスに対するコストを比較し，その優劣を評価する．続いて **4.2** では，系内の分散 VM 群が，近傍オブジェクトモデルに基づきオブジェクトを適応的に配置する場合に，系へ及ぼす負荷及び系全体から見た効率化の限界について評価する．これらの評価は，シミュレーションモデルにより行う．

4.1 近傍オブジェクトモデルの有効性

本節では，複数の分散 VM /オブジェクト/利用者が存在する系を対象に，分散 VM 群の協調によるオブジェクトの適応的配置をめざす近傍オブジェクトモデルの有効性について検証する．

分散 VM を用いて近傍オブジェクトモデルを実現する系として，VM を $N \times N$ の格子状に接続した図 1 で示すネットワークを作成する．現実のネットワークでは，多数のノードが多様なトポロジーで相互に接続されているが，ここでは，それを近似的にモデル化する例として格子状の接続を用いた．以下では，格子状ネットワークの座標 (x, y) に位置する VM を vm[x][y] と表記する．系内の利用者については，個々の利用者をモデル化する代わりに，オブジェクトアクセスがある確率に従って発生するというレベルの抽象化を行う．また，系内のオブジェクトは，各 VM から自由にアクセスされる．

各 VM 間のリンクには，そのリンク上で通信に要する時間を抽象的に表す値としてリンクコストを定義した．インターネットにおける通信では，パケットの到着間隔やデータ転送時間，再送の発生などについてすその長い分布に従うとの報告がなされており [1], [7], [16]，この事実を反映させるため，各リンクコストもすその長い分布に基づいて生成した．また，VM 間の通信量は，主要な通信データにあげられるファイルやストリームのデータサイズと関連がある．筆者らが，UNIX 及び Windows における全ファイル（約 18.5 万ファイル）のサイズを実測したところ，上のようなすその長い分布を形成していることが判明した（図 2：両対数グラフ）．

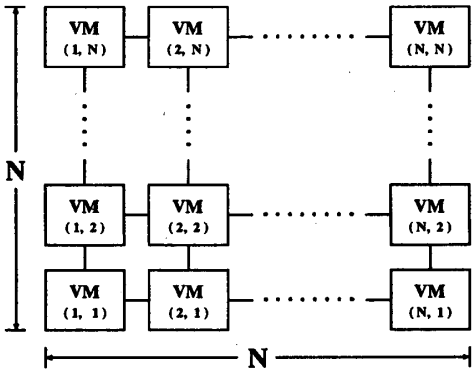


図 1　格子状 VM 網
Fig. 1　Grid-form VM network.

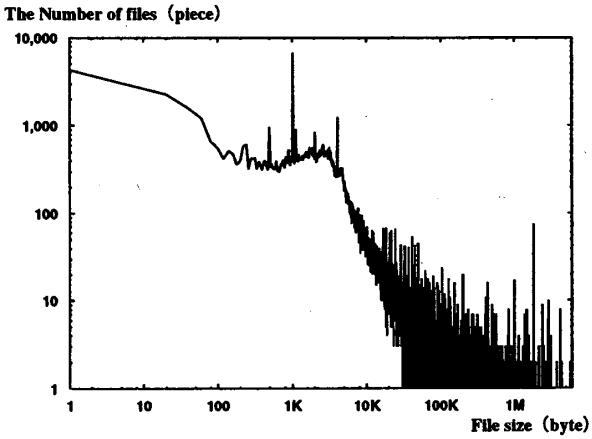


図 2　ファイルサイズの分布
Fig. 2　A distribution of file size.

更に，ストリームデータのシーン長 [5], [12] なども裾の長い分布に従うことから，リンクコストと同様，各アクセスごとの通信量もすその長い分布になるよう動的に決定した．

このような前提を備えた系において，vm[1][1]～vm[N][N] よりランダムに一つの VM を選択し，以下のアルゴリズムに基づいてオブジェクトの探索/移動を行う（これを 1 回の試行と定義し，$N \times N$ 回の試行を 1 サイクルと定義する）．

（I） すべての分散VMでは，各サイクルごとに確率 p でオブジェクトアクセスが発生する．

（II） オブジェクトアクセスが発生したVM (vm[X][Y]) では，該当するオブジェクトを求めて，下記の疑似コードにより近傍のVMを探索する．

```
for (i = 1; i ≦ N; i++) {
  for (x = X − i; x ≦ X + i; x++) {
    for (y = Y − i; y ≦ Y + i; y++) {
      if (vm[x][y] 上にオブジェクトが存在する) {
        return(vm[x][y]);
      } } } }
```

（III） 探索の結果，vm[x][y] 上で発見したオブジェクト $O_{(x,y)}$ が以下の二つの条件を満たす場合は，近傍外に存在すると判断して探索元VM (vm[X][Y]) まで移動させる．

(a) vm[X][Y]～vm[x][y] 間の通信コスト $\geqq C_n$

(b) $O_{(x,y)}$ の通信量 $\geqq V_c$ (2)

ただし，C_n, V_c は近傍を判断するためのしきい値

C_n：通信コストに対するしきい値

V_c： 通信量に対するしきい値

各VMはランダムな順番で1サイクル中に1回だけオブジェクトの探索/移動を行う（1サイクル中に2回以上オブジェクトの探索/移動を行うVMは存在しない）．1回のシミュレーションは10サイクル行う．シミュレーションの開始時点では，系内のオブジェクトは格子状ネットワークのほぼ中心に位置するVM (vm[N/2][N/2]) 上にのみ存在させる．

以後の議論では，オブジェクト間の通信に要する時間を抽象的に表す値として通信コストを定義する．今回のシミュレーションでは，オブジェクト間を結ぶ通信経路に沿ったリンクコストの総和を計算し，これを通信コストとした(注1)．通信コストは，例えばリンク情報の収集間隔を密にすることで，リンク情報に変動があっても比較的正確に見積もることができる．一方，オブジェクトの通信量はアクセスごとに変動し，これを事前に見積もることは難しい．今回は式 (1) の代わりに，vm[X][Y] が通信量を計測してしきい値 V_c を超えた時点で移動を行わせた．また，近傍の判断に各VMの負荷を利用するには，オブジェクトの処理負荷を予測する必要がある．これは，オブジェクトの通信量に対する予測と本質的に類似しており，今回は通信量のみを条件に選んだ．

更に，実際の環境を考えると，1サイクル中に一つのVMが複数のオブジェクトにアクセスする，あるいは複数のVMが同じオブジェクトにアクセスする可能性があげられる．前者については，種類の異なるオブジェクトが連携して動作する場合を検討する必要がある．しかし，実際に連携するオブジェクトを事前に確定することはできないため（膨大な候補をあげることはできるが），最終的な決定は実行時において動的に行わざるをえない．これより，複数のオブジェクトによる連携まで考慮した近傍の判断は，可能性のあるオブジェクトを適当な基準に従い対象に加えても，情報量が増える一方で精度の向上はあまり期待できない．したがって，近傍は個々のオブジェクトごとに独立して判断し，種類の異なるオブジェクトの存在は近傍の判断に影響を及ぼさないとした．これに対し，オブジェクト間の連携によりオブジェクトアクセスの発生する確率は変化する．しかし，最終的に他の（種類の異なる）オブジェクトが及ぼす影響は，各VM上で1サイクル中にオブジェクトアクセスの発生する確率 p の変動へと集約できる．個々のオブジェクトアクセスは基本的に独立した事象であることから，p の変動量及び変動時間もVMごとに独立である．今回は，各試行ごとに2～10の整数を一つ選び（これを n とする），$p = 1/n$ としてランダムに変化させた．

また，後者（複数のVMが同じオブジェクトにアクセスする）については，同じオブジェクトを短い時間内に取り合うことでオブジェクトの振動が生じる場合を検討する必要がある．これは，オブジェクトに対してロック機構を導入することにより防ぐことができる．VMは，発見したオブジェクトがロックされていれば（既に他のVMにより使用されていれば），ロックされていない次のオブジェクトを求めて探索を再開する．ここで導入したロック機構は，VM間の完全な排他をめざすものではなく，オブジェクトの振動を鎮静化することが目的である．したがって，系全体でデッドロックが発生しないように，効率を落とさない範囲でロックをタイムアウトさせる必要がある．ロック機構の導入は，オブジェクトの総数がロックされたオブジェクトの数だけ減少したとみなすことができる．

以上のシミュレーションにおいて，

［ケース1］ 近傍に応じて適応的にオブジェクトを移動させた場合

(注1)：インターネット上の通信では，必ずしも最適な通信経路が選択されるわけではない．今回のシミュレーションにおいても，これを反映するために通信コストが最小となる通信経路の決定などは行わない．

表 1 アクセスコストの比較
Table 1 A comparison of the access cost.

アクセス方式	総通信コスト	総移動コスト
ケース 1	3.77×10^7	7.52×10^5
ケース 2	0	2.66×10^6
ケース 3	2.10×10^8	0

［ケース 2］ 常にオブジェクトを近接させた場合

［ケース 3］ 全く移動を行わない場合

の 3 種類のケースに対し，通信コスト及び移動コストの比較を行った．移動コストは，通信コストと同じく移動に要する時間を抽象的に表す値である．条件式 (2) におけるしきい値は，それぞれ $C_n = 100$ 及び $V_c = 10$ とし，系の規模は $N = 100$（分散 VM の総数 = 10,000 台）とした．表 1 にその結果を示す．

インターネットを介して通信を行う場合，散発的な通信は途中経路のルータにおける経路情報キャッシュの状況などにより，通信効率が下がる傾向にある．一方，連続して通信を行う場合（オブジェクトを移動する場合など）は，キャッシュの効果により通信効率が上がる．筆者らがインターネット上で実測した値では，両者の差は最大で 100 倍程度あった．オブジェクトを UNIX のファイルに対応させると，その平均サイズは約 90 kByte であり，総通信量をオブジェクトサイズの 1/100 と仮定すれば，通信効率の違いにより両者の通信所要時間はその差が縮まるため，インターネット環境における移動時間と総通信時間の最終的な比は，たかだか数十倍程度と見積もれる(注2)．アクセスコストをアクセスに要する時間を表す値と考えると，適応的に配置する方式は他の方式に比べて数倍の効率（アクセス時間の短縮）を示した．

次に，リンクコストの分布を変更し，ふくそうしたリンク（コストの大きいリンク）の比率を増やした状態で，同様のシミュレーションを行った．表 2 にその結果を示す．表 1 と比べてふくそうの影響を受けることにより，適応的配置の効率は低下している．

ふくそうが発生すると，式 (2) における通信コストの条件が相対的に厳しくなり近傍の範囲は狭くなる．その結果，オブジェクトの移動頻度が上がるため，オブジェクトアクセスは適応的な配置による方式よりケース 2 の方式に近くなると予想される．これを検証するため，ふくそうの発生が少ない環境で式 (2) のしきい値を変化させながら測定したアクセスコストの変化を図 3（等高線表示）へ示す（以下では，移動コストを通信コストの 10 倍と仮定して評価を行った）．

表 2 ふくそう時におけるアクセスコストの比較
Table 2 A comparison of the access cost in a congestion situation.

アクセス方式	総通信コスト	総移動コスト
ケース 1	1.39×10^8	3.59×10^6
ケース 2	0	9.29×10^6
ケース 3	3.83×10^8	0

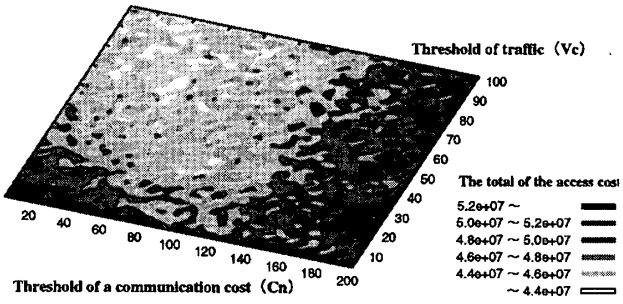


図 3 しきい値の変化に伴うアクセスコストの変化
Fig. 3 A transition of the access cost according to the transition of threshold.

しきい値の条件が厳しい領域（C_n, V_c がともに小さい領域）は近傍の範囲が狭い場合に該当し，ケース 2 と同じくアクセスコストも高いことがわかる．しきい値の条件を緩めるにつれ，式 (2) を満たすノードが増えることから近傍の範囲も広くなる．この場合，オブジェクトアクセスはケース 3 の方式に近くなり，そのコストも次第に高くなることがわかる．また，しきい値として適当な値を選択した領域はアクセスコストも低く，適応的な配置による効率化が期待できる．

以上より，適応的な配置による効率の向上には，

- ふくそうの発生したリンクを回避すること
- ネットワーク内にふくそうを発生させないこと

が重要といえる．

また一方で，適応的な配置の効率は，系内のオブジェクト数を増やすことにより向上するとも予想される．表 3 は，ふくそうの発生が少ない状態でオブジェクト数だけを変化させ，適応的配置によるアクセスコストを比較した結果である．

この結果では，オブジェクト数の増加により，適応的な配置の効率は改善されていることがわかる．次節では，オブジェクト数の増加と適応的配置の効率との関係，及びオブジェクト数の増加が系に与える影響について評価を行う．

(注2)：例えば，オブジェクトのサイズを 100 kByte，総通信量を 1 kByte とすると通信時間の比は 100:1 になるが，通信効率が 5 倍異なると通信時間の差は 20 倍に縮まる．

表 3　オブジェクト数の違いによるアクセスコストの比較
Table 3　A comparison of the access cost according to the difference in the number of objects.

オブジェクト数	総通信コスト	総移動コスト
5	2.07×10^8	5.33×10^6
100	1.39×10^8	3.59×10^6

4.2　近傍オブジェクトモデルが環境へ及ぼす影響

近傍オブジェクトモデルを適用した場合，オブジェクトアクセスの条件に応じてオブジェクトが移動し，適応的に配置が変わることから，その実在位置には偏りが生じる．オブジェクトの利用者は，自身の効率化をめざして必要なオブジェクトを手近な場所へ配置することから，オブジェクトの総数に対して利用者が多い場合は，オブジェクトの移動頻度が大きくなる．インターネットに代表される非一様なネットワーク環境では一般に通信コストが高いため，オブジェクトが環境内を激しく移動する状況は，逆にオブジェクトアクセスの効率が低下する．これより，近傍オブジェクトモデルの適用が系に及ぼす負荷は，オブジェクトの移動頻度を尺度として計測できる．オブジェクトの移動頻度は，利用者が系内に配置されたオブジェクトを任意に選択できるという前提のもとで，系内のオブジェクト数 O_n を増やすことによりその低減が期待できる．本節では，オブジェクト及びその利用者数とオブジェクトの移動頻度との関係をシミュレーションにより評価し，系全体から見た効率化の限界について検証する．

分散 VM の数に比べてオブジェクトの数が十分にある場合では，探索されたオブジェクトは，近傍の VM による探索で手近な場所に配置されたオブジェクトの可能性が高くなる．この場合，条件式 (2) より通信コストの低い（しきい値 C_n 以下の）リモートアクセスとして処理できるため，オブジェクトの移動は生じない．つまり，オブジェクトが移動するかどうかの確率は，それ以前の近傍の動作に影響されることから，オブジェクトの動作は単純な二次元のランダムウォークとしてモデル化できない [17]．以下では，前節と同じシミュレーション環境においてオブジェクトの移動頻度を計測する．

まずは，オブジェクトの移動状況を調べるため，系内に存在するオブジェクトの一つに着目し（以下，サンプルオブジェクトと呼ぶ），サンプルオブジェクトの移動を追跡する．図 1 の格子網を構成する各分散 VM 上でランダムにオブジェクトアクセスを発生させ，

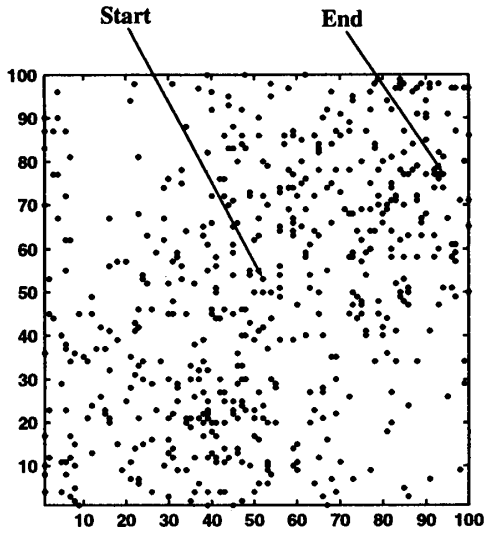


図 4　サンプルオブジェクトの分布（O_n：5 個）
Fig. 4　A distribution of a sample object (O_n : 5).

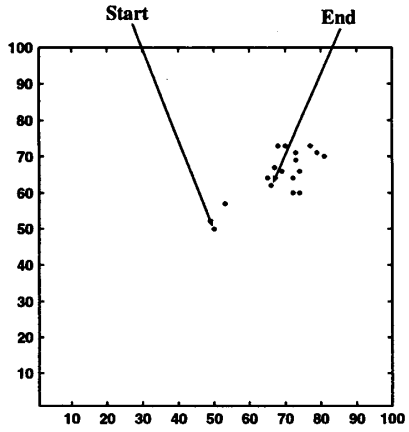


図 5　サンプルオブジェクトの分布（O_n：40 個）
Fig. 5　A distribution of a sample object (O_n : 40).

前述の疑似コードに従い探索→移動を行わせながら，サンプルオブジェクトが存在した VM の位置を記録した．その結果を，図 4（O_n：5 個）及び図 5（O_n：40 個）に示す．両図は図 1 を縮小した図であり，その各座標は図 1 で示した分散 VM の一つに該当する．図 4，図 5 内の黒点で表された座標は，サンプルオブジェクトが存在したことのある VM の位置を示す．これは，ある時刻にサンプルオブジェクトが黒点で示される VM 上のどれかに存在することを意味しており，両図はサンプルオブジェクトの存在分布図といえる．

系内のオブジェクト数が少ない場合は，分散 VM どうしがオブジェクトを取り合うため，オブジェクトの移動が激しくなる（図 4）．

一方，オブジェクト数が多い場合は，時間の経過に伴い各 VM の周囲に必要なオブジェクトが存在する確

率も上がる．よって，オブジェクトを移動させずとも効率的なアクセスを行えることから，その移動はおさえられる（図 5）．

次に，系内のオブジェクト数とその移動頻度との関係を調べるため，上記と同じ系でオブジェクト数 O_n を 1〜60 個まで増やしながら，各場合ごとにサンプルオブジェクトの移動頻度を計測した．系内にオブジェクトが n 個ある場合，理想状態(注3)では，オブジェクト一つ当りの移動頻度は

$$O(1/n) \tag{3}$$

に従うとみなせる．しかし，オブジェクトが多くなるにつれ，それ以前の探索で得られた近傍オブジェクトの存在が影響を及ぼし，オブジェクトの分布に偏りが生じるため，オブジェクト一つ当りの移動頻度も式 (3) からのかい離が大きくなる．

各オブジェクト数ごとの移動頻度を，図 6（両対数グラフ）に示す．移動頻度は特徴的に変化し，次の三つの領域に分けることができた．

領域 I：系内のオブジェクト数 O_n が増加するにつれ，サンプルオブジェクトの移動頻度もほぼ式 (3) に従って低下する領域

領域 II：同，サンプルオブジェクトの移動頻度が式 (3) から大きくかい離する領域

領域 III：同，サンプルオブジェクトは移動を行わない領域

領域 I / II の境界は，図 6 のグラフより小区間 $[\, O_n - d,\ O_n + d\,]$ における回帰式の傾きを求め，O_n を変化させながら傾きの変化を調べることにより決定できる．今回の解析では，$d = 6$ とした場合に，$O_n = 26$ の前後で回帰式の傾きが -0.51 から -4.26 へと大きく変化した．

本論文では，系内を移動しないオブジェクトが出現する領域 II / III の境界を，近傍オブジェクトの飽和数と定義する．他の条件を変えて同様のシミュレーションを行っても，グラフはおおむね三つの領域に分けることができた．図 7（両対数グラフ）に系内の総ノード数 n を 4〜101,124 個（図 1 における $N = 2 \sim 318$）まで増やしながら，各場合ごとにその飽和数を集計した結果を示す．近傍オブジェクトの飽和数 (O_{lim}) は，ほぼ直線上に並んでおり，その回帰式は式 (4) となる．

$$O_{lim} = e^{-1.48} \cdot n^{0.56} \tag{4}$$

$$(\log O_{lim} = a \cdot \log n + b,\ a = 0.56,\ b = -1.48)$$

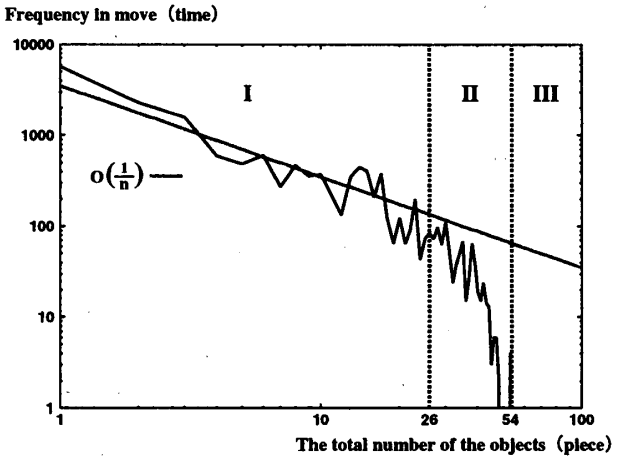


図 6　オブジェクト数と移動頻度の比較
Fig. 6　The comparison with the number of objects and a frequency in move.

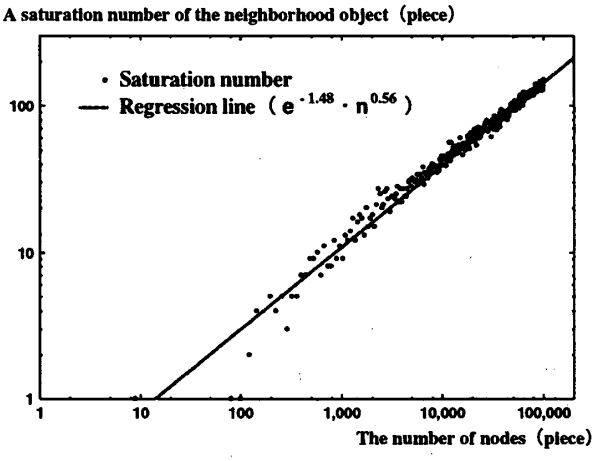


図 7　飽和数の推移
Fig. 7　Transit of the saturation number.

以上の評価結果より，近傍オブジェクトモデルでは，系内のオブジェクト数を増加させてもある基準を越えると移動しないオブジェクトが発生するため，オブジェクト数の増加による効率化には限界がある．一方，飽和数以下の範囲で複製を用意すればオブジェクトアクセスの効率を向上できるため，飽和数を見積もることにより系全体の最適化を図ることができる．

5. むすび

本論文では，分散プログラミングにおける実行環境の不均質性を近傍という形でモデル化して扱い，以下のアイデアにより透過性の提供と実行環境の最適化を両立する方式を提案した．

(注3)：各オブジェクトが系を n 等分した部分系へ均等に配置され，部分系内でのみ移動を行う状態．

- 分散仮想マシン (VM：Virtual Machine) を導入し，VM 全体で共通の分散実行環境を実現する．
- 各分散 VM では，物理的な実行環境に基づいた近傍を認識する．
- オブジェクトへのアクセス時は，分散 VM 間で近傍を意識した動的なオブジェクト探索を行う．

更に本論文では，近傍オブジェクトモデルを適用した場合の有効性について，シミュレーションによる評価を行った．

近傍オブジェクトモデルでは，系内に存在するオブジェクトの数を一定範囲で増加させることにより，オブジェクトの移動をおさえながら全体の通信コストを低減できる．しかし，ネットワークにふくそうが発生すると，その効率は低下する．したがって，近傍オブジェクトモデルを有効に適用するには，ふくそうを回避する，あるいはふくそうを発生させないことが重要になる．現在は，上記の成果より，オブジェクトが移動した場合の型情報の管理や近傍オブジェクトモデルを実現する言語の設計/処理系の開発などについて研究を進めている段階である．

文　献

[1] M.E. Crovella and A. Bestavros, "Self-similarity in world-wide-web traffic: Evidence and possible causes," Proc. of SIGMETRICS'96, pp.160–169, 1996.

[2] N. Fujinami and Y. Yokote, "Naming and Addressing of Objects without Unique Identifiers," Sony Computer Science Laboratories Technical Report, SCSL-TR-92-004, 1992.

[3] S. Hirano, "HORB: a distributed object oriented language for worldwide programming," Proc. of WOOC'96, 1996.

[4] S. Hirano, "Constructing a portable ORB for ditributed execution of Java," SWoPP'96 IPSJ Technical Report, 96-OS-73, pp.55–60, 1996.

[5] K. Ishibashi, T. Kimura, and T. Ozawa, "A measurement-based performance evaluation method for IP network and its implementation," Proc. of 11th ITC Specialist Seminar, pp.272–277, 1998.

[6] Object Management Group, "Common Object Request Broker Architecture (CORBA/IIOP)," http://www.omg.org.com/

[7] V. Paxson and S. Floyd, "Wide-area traffic: the failure of Poisson modeling," IEEE/ACM Trans. Networking, vol.3, pp.226–244, 1995.

[8] Sun Microsystems Inc., "Java Computing," http://java.sun.com/java/

[9] Sun Microsystems Inc., "Remote Objects for Java (tm)," http://splash.javasoft.com /pages/intro.html

[10] A.S. Tanenbaum and R. van Renesse, "Distributed Operating Systems, ACM Computing Surveys," vol.17, no.4, pp.419–470, 1985.

[11] A.S. Tanenbaum and M. van Steen, Distributed Systems Principles and Paradigms, Prentice-Hall, Inc., 2002.

[12] W. Willinger, M.S. Taqqu, R. Sherman, and D.V. Wilson, "Self-similarity through high-variability: Statistical analysis of Ehternet LAN traffic at the source level," IEEE/ACM Trans. Networking, vol.5, no.1, pp.71–86, 1997.

[13] Z. Yang and K. Duddy, "CORBA: A Platform for Distributed Object Computing," Operating Systems Rev., vol.30, no.4, pp.4–31, 1996.

[14] Y. Yokote, "The New Mechanism for Object-Oriented System Programming," Sony Computer Science Laboratories Technical Report, SCSL-TR-92-004, 1992.

[15] 田井秀樹，山本　学，"移動エージェント技術の現状と今後の課題，" コンピュータソフトウェア，vol.16, no.5, pp.2–13, 1999.

[16] 高安秀樹，フラクタル，朝倉書店，1986.

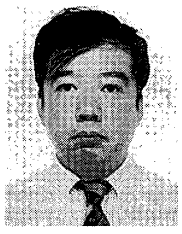
[17] 森村英典，確率・統計，朝倉書店，1974.

[18] 横手靖彦，"オブジェクト指向分散オペレーティングシステム，" コンピュータソフトウェア，vol.9, no.3, pp.15–35, 1992.

（平成 14 年 12 月 9 日受付，15 年 3 月 14 日再受付）

地引　昌弘　（正員）

平 4 東工大大学院理工学研究科情報科学専攻博士前期課程了．同年日本電気（株）入社．平 15 筑波大大学院経営・政策科学研究科企業科学専攻博士後期課程了．現在，NEC ネットワーク開発研究本部に勤務．分散アーキテクチャ，ソフトウェア，通信システム，経路制御などの研究に従事．博士（システムズ・マネジメント）．情報処理学会会員．



久野　靖

昭 59 東工大大学院理工学研究科情報科学専攻博士後期課程単位取得退学．同年同大理学部情報科学科助手．筑波大大学院経営システム科学専攻講師，助教授を経て，現在同教授．プログラミング言語，プログラミング環境，ユーザインタフェース，情報教育に興味をもつ．理博．著書「UNIX による計算機科学入門」（丸善），「入門 JavaScript」（アスキー）など．情報処理学会，日本ソフトウェア科学会，ACM，IEEE Computer Society 各会員．