オペーレーティング・システム 54－1
（１９９２. ３. １３）

ポリシーに基づく適応型ＰＣの実現

神原 順一
東洋大学　大学院　工学研究科

年々ネットワークの規模は拡大し、ネットワーク上の資源や環境を管理している。　これらを管理しているノードと、そうでないノードでは負荷が偏っている。　そこで本研究では分散環境においてユーザからの方針とシステムにおける方針と判定することによって、タスク内の通常の関数呼び出し（ＬＰＣ）と、ネットワークを介した遠隔関数呼び出し（ＲＰＣ）のいづれかを提供する。

Adaptive Procedure Call Based on User and System Requirements

Junichi Kambara

Department of Infomation and Computer Sciences
TOYO UNIVERSITY

2100 Nakanodai, Kawagoe, 350 JAPAN

Recently, the scale of networks is increased year by year, manageing of network resources and environments becomes so difficult.
This features better remote execution capability coordinated by the broker task for the user side and system side requirements. The agent selects suitable remote execution methods whether Remote Purocedure Call(RPC) or Local Procedure Call(LPC).

1　はじめに

近年、ＬＡＮがめまぐるしく発展して、ネットワーク上に数百台以上のＵＮＩＸ　ＷＳが接続されるようになった。　このような状況では、ファイルやプリンタなどが共有され、有効的に活用されている。　しかし、現段階においては、このように便利な環境を支えているのは、数人のシスオペと、数台のＷＳである。　これらのＷＳは様々な資源を管理したり、ユーザの酷使にも耐えなければならない。　このような状況において、同じネットワーク上に存在しながら、資源を管理するＷＳと、管理しないＷＳの間では、明らかに負荷のかかり方が大きく異なる[1,2]。　そこで本研究では、分散管理された情報をもとに、それらを効率よく処理するための適応型ＰＣを提案する。

本システムは、分散ＯＳであるＭＡＣＨを利用し、そのＭＡＣＨが持つメッセージ機能とスタブ・ジェネレータＭＩＧを用いて、分散環境における関数を、その関数にまつわる要因に基づいて、その関数をそのタスク内の関数としての通常の呼び出し（以下、ＬＰＣと省略）と、ネットワークを介した遠隔呼び出し（ＲＰＣ）のいずれにするかを決定するシステムを構築する。　このようにＬＰＣとＲＰＣが同一のインタフェースを持つことにより、ユーザはネットワークを気にすることなくプログラミングすることができる。　また、ＭＩＧより高機能なスタブ・ジェネレータとしてＮＣＳ[3]などがある。　さらに、どのように処理するかを決定するために、ユーザの方針（ポリシー）とシステムの方針の双方を比較しながらシステムが、その方針を決定する。　このためユーザの考えも考慮されるので、ユーザの意に反するような方針を抑制することができる。

2　システム構成

本システムは、タスク内において、ほかのタスクとのインタフェースをつかさどるエージェント・モジュール（以下、エージェントと省略）と、それぞれのノードにおいて、ＲＰＣと情報の管理を行うアシスタント・タスク（以下、アシスタントと省略）と、複数のアシスタントから構成されるアシスタント・ドメインの内において、各アシスタントからの情報を一括し集中的に管理するブローカ・タスク（以下、ブローカと省略）から構成される。　これらの３つのタスクがそれぞれ協調することによって、与えられた仕事をが定義されている関数（以下、ジョブ関数と省略）の適応型ＰＣ（プロシージャ・コール）が実現されている。

2．1　エージェント

エージェントは、サービスを受けるすべてのタスク内に存在し、そのサービスによって処理しようとするジョブ関数に対し、そのジョブ関数を協調して行う複数のアシスタントとの間で、ジョブ関数呼び出しや必要となる情報のやり取りを行うことによって、そのジョブ関数の処理を行う。　このときエージェントは、以下に示すような機能が必要である。

ブローカ・インタフェース機能

エージェントは与えられたジョブ関数を補佐してくれるアシスタントの存在をブローカから得る必要がある。　このとき、そのジョブ関数が必要としている情報を保持しているアシスタントも得なければならない。　これらの情報は、ジョブ関数を行うためには必要なものであり、かつ、このジョブ関数をどのように構成するかを判定するために必要な情報でもある。

方針判定機能

上記の機能によって、このジョブ関数をこの環境においてどのような構成にするか、その方針を決定する機能である。　この機能で提供される方針については、後章で述べる。

関数呼び出しの異化（仮想化）機能

この機能は、いわゆるＲＰＣのスタブの機能であり、ＭＡＣＨのＭＩＧの生成するスタブを利用し、ＭＡＣＨ　ＯＳ上での移植を容易にする。

２．２　アシスタント

アシスタントは、エージェントから依頼された関数や、情報の受け渡しなどを行うサーバ・タスクである。　しかも、このタスクは各ノードに存在していて、このシステムが方針を決定する上で必要とする要因を収集し管理している。

要因管理機能

ここで収集される要因は、このシステムが方針を決定する上で欠くことのできない要素である。　アシスタントはこの要因を定期的に収集し、管理する機能を持つ。

ブローカ・インタフェース機能

上記の機能で収集された要因の中で、システムが必要とする情報をブローカに登録する機能である。　このときこの機能を最適化しないと、不用意にネットワーク・トラフィックを上げてしまうことになる。　したがって、ブローカ・インタフェースは、収集されたすべての要因を一度登録要求を出してしまうと、次回からは必要最低限の要因のみの登録要求を行う。

アシスタント機能

エージェントから送られてきた関数を、同化し処理を行う機能である。　しかし、本研究において、この機能はスタティック・リンクを用いて、すべてのジョブ関数をアシスタントが所有する形になっている。　本来この機能は、アシスタントから送られてきた要求に基づいて、ジョブ関数ライブラリをダイナミック・リンクすることによって実現べき機能である。

情報管理機能

アシスタントは、このシステムで必要な情報に対するロケーション情報を管理する機能である。　この中には、エージェントから送られてくる情報のロックとその解除を行う機能も含まれる。　また保持している情報をほかのタスクに複製することも行う機能を有している。

２．３　ブローカ

ブローカは、アシスタントが収集した要因を記憶管理し、エージェントがその参照を求めてきた場合に、管理している要因と必要な情報のロケーション情報をエージェントに返答するタスクである。　すなわち、このシステムにおけるインフォメーション・サーバである。

アシスタント・インタフェース機能

アシスタントとのインタフェースを管理する機能である。　しかし、ブローカは受動的にアシスタントからの情報を収集するので、不特定多数のアシスタントの要求を受けることになる。

エージェント・インタフェース機能

エージェントからの要求を受ける機能で、その要求を満たすアシスタントとその要因を検索し、エージェントに返す機能を有する。

要因管理機能

アシスタントから送られてきた要因を管理する機能である。　このときアシスタントから送られてくる情報は、縮小化されている場合もあるので、そのときには既存の情報と補完も行う。

3　方針

方針には、大きく分類すると、ユーザがそのシステムに求める方針とシステム開発者が定義する方針に分類することができる。

前者（すなわちユーザ型方針）とは、ユーザがシステムを利用する上で、もしくは現在使用していて、率直に肌で感じとることができる「もっと速く処理したい」とか、「もっと小さなコードにしたい」など、「速さ」や、「質量」などであり、ユーザの利用したときの実感に深く結びついているものである。　これに対して後者のシステム開発者の定義する方針（すなわちシステム型方針）とは、開発者がシステムを構成し、開発する上でのよりどころにした「概念」や、「法則」などであり、システム開発者の動機と深く結びついた方針である。　これはＬＯＣＵＳにおける問題指向型プロトコルの概念[4] と同じ考え方である。

これら２つの方針は、一定条件で結びついているものではなく、そのときの環境や状態においても耐えず変化しているものである。　たとえば、ユーザにとっては、そのような「概念」によって設計されているのかはある程度考慮されるが、それよりより「速く」、より「便利」に利用できるものなどが方針となる。

3．1　方針と要因

要因とは、システムが方針を決定する上で参照するタスクの状態や環境である。　ここでのタスクとは主にアシスタント・タスクのことで、これらの状態とユーザの示した方針を用いて、分散を行うための方針を決定する。　この要因には次のようなものが上げられる。

長期累積の負荷

アシスタントに対して、その負荷を長期にわたり調べて、そのアシスタントが存在するノードが、現在の環境において、積極的に処理を行っているかどうかを調べようとするものである。　したがって、この負荷が大きいということは、このノードが積極的に利用されていることを意味する。　このような場合、このノードにおいてはエージェントは消極的な方針をとる関数に対し、積極的にＲＰＣしようと試みる。

短期平均の負荷

アシスタントに対して、その負荷を短期的にまとめ、このノードの負荷が低い場合、分散協調型の方針に基づくアシスタントとして参加する。

保持している情報の質量

アシスタント内に保持する情報が多い場合、アシスタントはその情報を必要とする関数を積極的にサービスしようとする。　また、このようなアシスタントに対し、エージェントは情報集中型の方針を持つアシスタント、もしくは協調分散型のアシスタントと判定する。

記憶資源の許容量

アシスタントが持つ記憶資源の許容量が大きい場合、そのアシスタントは情報集中型のアシスタントとして、利用されることが多くなる。

3．2　方針と仮想化（異化）

関数と情報の仮想化を分離することによって、関数のみを持つタスクや、情報のみを持つタスク、関数と情報の双方を持つタスクのいずれもが矛盾なく共存することができる。　これらのタスクは、それぞれの状態において、潜在的にある方針を持っている。　例えば、関数のみを持つタスクは、その関数を行うときに必ず情報を同化してから処理を行うことになる。　このためこのタスクでは積極的に情報を得ようとする。　これはコンピュータ・サーバを仮想化したものであると言える。　また、情報のみを持つタスクは、その情報を必要

とするタスクに自分が保持している情報を送ることになる。　すなわち、積極的に自分の保持する情報を放出するのである。　これはストレージ・サーバを仮想化したものである。

これとは異なり、関数と情報の双方をもつタスクは、潜在的に存在する方針を得ることは困難であり、様々な要因と２つの方針を用いて、その動作を決定することが重要である。　まず、この関数に対して、このエージェントが積極的参加であるかどうかを知る必要がある。　この場合、積極的参加であれば、あまり多くのアシスタントを用いずに、なるべく情報をこのエージェントのあるタスクに集めようとする。　また、逆に消極的参加の場合、なるべくほかのタスクだけで処理を行おうとする。

ほかに、アシスタントや、エージェントをもタスクにおいて、行おうとする関数が必要とする情報の質量によっても、異なった解釈を行わなければならない。　このとき、情報を多く保持しているアシスタントは、情報集中型か分散協調型のタスクであると認識することができる。

３．３　情報集中型の方針

システム内のあるタスクの必要とする情報が、いろいろな記憶領域に分散されている場合、そのタスクの存在する記憶領域を中心に分散された情報を集めて、その処理を行うことを情報集中型の方針という。　この場合、情報を物理的に一か所に集め、そこでジョブ関数を行うので、一貫性が容易に保つことができる。　しかし、分散された情報を一か所に集めなければならないので、その情報の質量が大きい場合、相当の負荷を背負ってしまう欠点を持つ。　この場合必要としている情報を、集めようとしている記憶領域に送ってからジョブ関数を行う。　これらの基本動作を以下に述べる。

１）　情報のロケーションを得る。
与えられたジョブ関数が必要としている情報のロケーション・リストをブローカより入手する。

２）　ジョブ関数を行うタスクを決定する。
得られたロケーション・リストから、どのタスクに情報を集めるかを決定する。

３）　情報をロックする。
それぞれの情報を保持しているアシスタントに対してロックをかける。　また、指定されたタスクに保持している情報の複製を送る。

４）　ジョブ関数を行う。
情報が集められたタスクに対してジョブ関数を発行する。　このとき、ジョブ関数を発行しようとするタスクに情報が集められていれば、通常のＩＰＣが行われ、それ以外の場合、いわゆるＲＰＣを行う。

５）　情報のロックを解除する。
集められた情報は、ジョブ関数が終了した時点で書き戻されてから、ロックが外される。
しかしほかのタスクがその領域を必要としない限り、そのままキャッシングされる。

この一連の流れの中で重要な動作は、情報をどのタスクに集めるかを決定することである。　この動作は以下の状態に場合わけを行った。

関数最適化方針

・・・ジョブ関数を発行するタスクを対象

この方針は、ジョブ関数を引き受けるタスクが存在しない、もしくはジョブ関数を発行するタスクの負荷が平均的に低い場合に選択されるべきものである。　この場合情報はすべて、このタスクに集められる。

情報最適化方針

・・・情報を多く保持するタスクを対象

この方針は、必要とされる情報が偏ったタ

スクに多く保持されいる場合や、すべての情報を集めるために必要な領域があるタスクが存在する場合に選択される。

負荷最適化方針

・・・相対的に負荷の少ないタスクを対象

この方針は、情報を保持しているタスクの中に、相対的に負荷が少ないタスクが存在するときに選択される。　これは基本的に関数最適化方針と双璧をなす方針である。

3．4　協調分散型の方針

あるタスクが必要とする情報が、いろいろな記憶領域に分散されている場合、そのタスクの持つ関数を、情報を持つそれぞれのアシスタントに送ることにより、処理が行われる。　この場合、その情報を保持しているタスクがそのジョブ関数を処理することができることが前提になっているが、実際には処理することが困難である場合、その情報をほかのタスクに情報を複写することになる。

1）　情報のロケーションを得る。

与えられたジョブ関数が必要としている情報のロケーション・リストをブローカより入手する。

2）　補佐するアシスタントを決定する。

得られたロケーション・リストから、補佐可能なアシスタントを選択する。

3）　情報をロックする。

それぞれの情報を保持しているアシスタントに対して、その情報のロックを求める。また、ある情報を持つアシスタントが、負荷が大きいなどの要因によって、補佐することが困難になった場合、その情報をほかのタスクに複写する。

4）　ジョブ関数を行う。

それぞれのアシスタントに対して、ＲＰＣを発行する。　また、エージェントを含むタスクにも情報が存在する場合、もちろんＬＰＣを行う。

5）　情報のロックを解除する。

この操作は、ほぼ集中型と同じであるので省略する。

前項の情報集中型とは異なり、協調分散型の方針では、そこに参加しているアシスタントの動作が異なる。　協調分散型の方針では、関数を受理できないアシスタントが情報を持っている場合に、次の場合が考えられる。

関数最適化

エージェントのあるタスクにその情報を送る。　これによって、この情報が分散的にまとめられる。

記憶最適化

多く記憶できる領域を持つタスク、またはその情報を多く保持しているタスクに送る。

これは、先ほどの関数最適化と似ているが、この場合この情報は集中するようにまとめられていく。

負荷最適化

負荷の少ないタスクに送る。　この場合、負荷を的確に判断しなければ、ここで送られた情報は、一定のタスクに落ち着かずに分散環境内を飛び交ってしまうことになるので注意が必要である。

4　分業処理による成長と協調処理による成長

4．1　分業処理による成長

分業処理による成長とは、現在のあるタスクの環境において、何らかの要因で処理することが困難になったジョブ関数を、新たに生成したタスク、もしくは既存のタスクに対して、そのジョブ関数を委託することにより分散処理を実現するものである。　これは成長の元になるタスクを親とした従属的な成長であり、トップ・ダウン型の構成になる。　これらジョブ関数を委任されたタスクは、親のタスクを基本とするドメインを構成する。　また、これらのタスクは、そのジョブ関数の仕事量が継続的に少なくなった場合に、再び親タスクに戻される。　すなわち、このことは親タスクがいつかは必ず、もとの状態に戻ることが可能であることを意味する。

このような考え方に基づく成長の１つの例としては、動的メモリ割り当てとマルチスレッドのよるサーバの構成が上げられる。　この構成法では、サーバのコントロール・スレッドはクライアントからのリクエストメッセージを受けとることのみに専念し、その受けとった要求は、新たに生成されたスレッドによって処理が行われる。　このとき、それぞれのスレッドが干渉しないように、スレッドが必要とする情報や記憶領域は動的に割り当てられていることが多い。　そして、スレッドはクライアントからの要求を処理するために必要な数だけ生成され、その仕事が終了した時点で消去される。　すなわち、このサーバ内で、クライアントからの要求一つ一つに対し、スレッドを基本単位とした従属的成長が行われているのである。

4．2　協調処理による成長

協調処理による成長とは、何らかの要因で現在のあるタスクの環境だけでは処理することが困難であるもしくは不可能であるジョブ関数を、そのジョブ関数を行うことが可能であるタスクや、その環境を実現できるタスクと協調して処理を行うことにより分散処理を実現するものである。　これは成長元になるタスクとそのジョブ関数を請け負うタスクとの共同作業的な成長であり、ボトム・アップ型の構成になる。　これらのジョブ関数は、成長元のタスクで処理される同時に、協調するタスクにおいても処理される。　しかし、この場合ほかのタスクがどのような環境にあるのか、またどのような処理を行うことが可能であるかを知る必要がある。　このことはある程度、タスクの状態を管理している、もしくはその状態を保存しているタスクが存在しなければならないことを意味する。　すなわち、協調処理による成長を望むタスクは、ある程度タスクの状態を管理しているタスクから様々な情報を得て、それから相手となるタスクを選びだし、このジョブ関数を協調して処理を行うのである。

5　むすび

本研究において、ネットワーク上の多様な分散環境に適応できるようにユーザの方針とシステムの方針を用いて、動的に変化する環境において、最適なプロシージャ・コールを行うために必要なシステムの構築し、実験を行った。

当面の課題

現在、ジョブ関数は、アシスタント・タスクに直接スタティック・リンクされている。　しかし、これからの分散環境においては、ジョブ関数は、ダイナミック・リンクした方が、メモリの効率や移植性などが向上すると考えられる。

長期的な課題

本システムは、ＭＡＣＨ　ＯＳ上に実装したものである。　したがって、タスク間通信はすべてメッセージによって行われる。　したがって、情報をメッセージにパッケイジングするために、多くの負荷を必要としているので、効率のよい通信手段へ変更する必要があると思われる。

6　参考文献

[1] Sape Mullender. Distributed Systems. ACM Press 1989.

[2] 前川 守,所 真理雄,清水 謙多郎　編. 分散オペレーティングシステム -UNIXの次にくるもの-. 共立出版 1991.

[3] Mike Kong. Network Computing System Reference Manual. Prentice Hall, 1990.

[4] G.J. Popek and B.J. Walker. The LOCUS Distributed System Architecture. MIT Press, Cambridge Massachusetts, 1985.