プログラミング 14－6
（1997. 8. 21）

分散問題解決による共有資源の実現

濱地 弘樹* 楢崎 修二** 吉田 紀彦* 牛島 和夫*
* 九州大学大学院 システム情報科学研究科
** 九州工業大学 工学部

分散環境で資源を共有するうえで通信負荷の軽減による効率化が問題となり、分散共有メモリなどの研究分野でも様々な方式が提案されている。本稿では、資源共有の効率化のために分散問題解決の枠組を用いた共有資源の実現手法を提案する。この手法では、エージェント集合が各プロセッサにおける共有資源への参照・更新頻度の情報を集めて、その情報を基にその共有資源の複製の配置を動的に決定する。またこの複製の配置決定戦略を共有資源への参照・更新に付随するメタレベル処理として実現する。実験により従来方式との比較を行ない効率化を確かめており、その結果についても述べる。

Efficient resource sharing using a framework for distributed problem solving

Hiroki Hamachi* Shuji Narazaki** Norihiko Yoshida* Kazuo Ushijima*
* Graduate School of Information Science and Electrical Engineering,Kyushu University
** Department of Computer Engineering, Faculty of Engineering, Kyushu Institute of Technology

When a resource is shared among processors in a distributed environment , it is necessary to reduce communication overhead. Various mechanisms for this purpose have been proposed in such research areas as distributed shared memory. We propose an efficient resource sharing mechanism using a framework for distributed problem solving. Our mechanism is that agents dynimically decide the placement of copies of the shared resource based on the count that each processor refers to it. And we describe how to process the strategy to decide the placement of copies at metalevel. we also compare our proposed mechanism with a traditional one.

1 はじめに

一般に分散環境下で処理を実行するためにはプロセス間での資源（データ）の共有が必要となる。共有された資源の参照や更新のためには通信が必要となる。分散処理の効率を考えるうえで、通信コストは無視できない問題である。しかし、従来の共有資源の実現方法では各プロセッサにおける資源の使用頻度の動的な変化に完全には対応していないので処理効率の改善が困難といえる。

従来の共有資源の実現方法の利点や欠点から、共有資源の複製を所持させる適切なプロセッサを動的に決定することによって共有資源を操作する時の通信回数を減らすことができると考えられる。複製を所持させる適切なプロセッサを選ぶ際には各プロセッサにおける共有資源の参照頻度を知る必要がある。そこで本研究では、共有資源の複製の配置決定を分散問題解決の枠組を用いて行なうことにする。

分散問題解決において、問題解決プログラムは問題ごとに異なるがエージェントの協調戦略は共通性が高いので、プログラムの再利用性を考慮して問題解決プログラムと協調戦略部分とを分離して記述す

る方法が提案されている [YNU95]。その研究では分離記述した協調戦略の再利用を実現するための方法として、メタオブジェクトプロトコルを用いて協調処理をメタレベル計算として実現している。こうすることにより協調戦略がエージェントの持つ局所情報の変化に付随するメタレベル計算として起動される。これによりエージェントの協調戦略に関する処理を自動化できるので、協調戦略に関する処理を問題レベルのプログラムに記述する必要がなくなる。

本研究では分散問題解決の枠組を利用して共有資源の複製の配置決定を動的に行ない、その複製の配置決定戦略を共有資源の操作に付随するメタレベル計算として実現する。

2 従来の共有資源の実現方法

共有資源の参照・更新には通信が必要となるが、処理効率の改善のためには通信回数を削減することが考えられる。本研究で考えている分散システムはメッセージの放送機能を持たないものを仮定している。その理由は、放送機能を持つシステムは規模や地理的な範囲が限られているからである。放送機能を使わないので通信回数は通信対象数に比例することになる。また、各プロセッサにおける資源の参照頻度は急激に変化しないものと仮定している。本研究で取り扱う共有資源は強い一貫性の保持が必要なものなので、資源の値の更新時には資源のすべての複製に対するロックと書き込みが必要となる。

この章では、従来の共有資源の実現方法と通信回数の観点から見たその欠点について述べる。

2.1 中央サーバ方式

複数のプロセッサで資源を共有するための最も簡単な戦略は、資源の唯一の複製を所持する中央管理サーバを設置することである (図 1)。この方式では、資源を所持しないプロセッサは資源の参照・更新を行なう時は常に中央サーバに通信しなければならない。そのため資源を所持しないプロセッサが頻繁に資源にアクセスする場合は、通信回数が増えるので効率が悪くなる。また中央サーバに通信が集中すると中央サーバがボトルネックになる恐れがある。

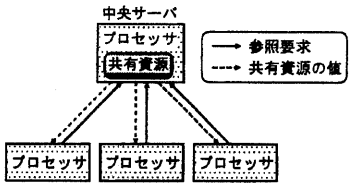

図 1: 中央サーバ方式

2.2 完全複製方式

中央サーバ方式では中央サーバ以外のプロセッサにおいて資源の参照・更新を行なう時には常に中央サーバへの通信を必要とした。そこで、各プロセッサに資源の複製を所持させることにより局所的に複製を参照できるので、参照時には通信を必要としなくなる (図 2)。また局所的に参照できるので、複数のプロセッサにおいて同時に資源の参照を行なえる。しかし複製の一貫性保持のために、資源の値を更新する時はすべての複製をロックして新たな値を書き込む必要があるので通信回数が増える。しかし、更新時の通信回数の増加よりも局所的に参照することによる通信回数の減少の方が影響が大きければ、複製を所持させた方がいいといえる。

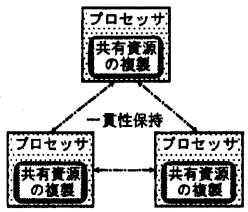

図 2: 完全複製方式

2.3 読み取り複製方式

分散共有メモリで一般的に使われているのは、資源の参照時に複製を作成する方式である。複製不所持プロセッサは資源の参照を行なう時に、複製所持プロセッサに通信して資源の値を受けとる。その時に複製を作成し、次回の参照時からは局所的に複製

を参照できるようになる。資源の更新を行なう時は、更新を行なうプロセッサが複製を持つ他のすべてのプロセッサに対して複製を破棄するように通信する。つまり、資源の参照を行なう時に複製を作成し、更新の度に複製の数を1つに戻す方式である。

この方式だと更新が起こるまでに1回でも参照したプロセッサは複製を所持するので、更新時に行なわれる通信が多くなるといえる。

2.4 従来方式のまとめ

ここまでに述べた従来方式の特徴をまとめる。複数のプロセッサにおいて共有資源が頻繁に参照されるようであればそれらのプロセッサに複製を所持させた方が全体として通信回数が削減され効率がよくなると言える。また、各プロセッサにおける資源の参照頻度が動的に変化した場合にも対応する必要があるといえる。

3 分散問題解決による共有資源の実現方法

2章で述べた従来方式の特徴を総合して判断すると、各プロセッサにおける共有資源の参照頻度に応じて複製の配置を動的に決定することで、効率のよい資源の共有ができると思われる。そのために、本研究では分散問題解決の枠組を利用して共有資源を実現する。

3.1 動作概略

本研究で提案する共有資源の実現方法の概略を図3に示す。各プロセッサにそれぞれ1つのエージェントを配置して、それらのエージェントの集合として共有資源を表現する。ここに出てくるエージェントのことを共有資源エージェントと呼ぶことにする。共有資源エージェントは各プロセッサにおける共有資源の参照頻度の情報を基に通信回数を小さくするような効率のよい複製の配置を決定する。プロセッサは共有資源を参照・更新したい時には自分のところの共有資源エージェントを参照・更新する。この部分はメタオブジェクトプロトコルを使用して実装している。詳細は4章で述べる。複数の資源を共有する場合は、各資源毎にそれぞれ共有資源エージェントの集合を用意することになる。

他の共有資源エージェントとは区別される特別なエージェントを1つ設ける。これを管理エージェントと呼ぶことにする。分散問題解決においては全体を管理する管理エージェントなるものは本来存在しないのだが、研究の第一歩として管理エージェントを設ける方法で共有資源の実現を行なうことにする。今後の課題として管理エージェントの処理の分散化を考えている。管理エージェントは共有資源の複製を常に所持し、その他の共有資源エージェントに複製を所持させたり複製を破棄させたりする決定を行なう。

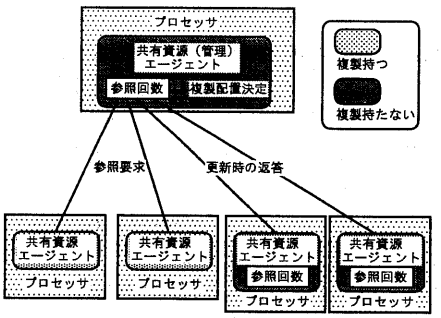

図 3: 分散問題解決による共有資源の実現

次に共有資源エージェントがプロセッサから参照・更新された時の処理を説明する。

資源の参照時の処理を図4に示す。共有資源エージェントはプロセッサから参照された時に、複製を所持しているなら局所的に複製を参照してその値を返す。その際、自分がプロセッサから参照された回数を数えておく。複製を所持していない場合は管理エージェントに通信して資源の値を要求する。その要求を受けた管理エージェントはそのエージェントが参照要求を送信してきた回数（＝プロセッサから参照された回数）を統計しておき、資源の値を返答する。エージェントはその返答を受けて資源の値をプロセッサに返すことになる。その時、読み取り複製方式と違って、複製は作らない。今述べたように、複製不所持エージェントがプロセッサから参照され

た回数を管理エージェントは常に知っている。

1回の参照につき、複製不所持エージェントに限り2回の通信が行なわれることになる。複製所持エージェントは参照時に通信を必要としない。

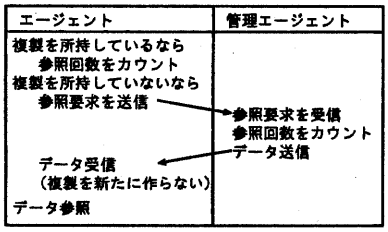

図 4: 参照時の処理

次に、資源の更新時の処理を図5に示す。共有資源エージェントは、プロセッサから更新された時に管理エージェントへ更新要求を送信する。その時、複製を所持していれば、局所的に複製を参照した回数も添付して送信する。管理エージェントは要求を受信すると、各エージェントの参照回数を基に複製の配置を決定する。その戦略については次節で述べる。複製所持エージェントに複製を破棄させたり、複製所持エージェントの複製の値を新しい値に書き換えたり、複製不所持エージェントに新たに複製を持たせたりするために、管理エージェントは各エージェントに通信する。いずれにせよ複製所持エージェントに対しては必ず通信することになる。複製所持エージェントはその返答を送る際に局所的に複製を参照した回数を添付して送信する。管理エージェントはすべての返答を得ることによって、複製所持エージェントがプロセッサに参照された回数を知ることができる。最後に、管理エージェントは更新要求を送信してきたエージェントに更新完了を伝える。このように更新が行なわれる時に複製の新しい配置を決定する。

更新時には、上で述べた処理の前後にすべての複製のロック・アンロックのための通信（それぞれ往復）が必要となる。それも踏まえると、更新時には1つの複製当たり6回の通信が必要ということになる。

これまでに述べたように、資源の更新が完了する時に管理エージェントはそれまでに各エージェント

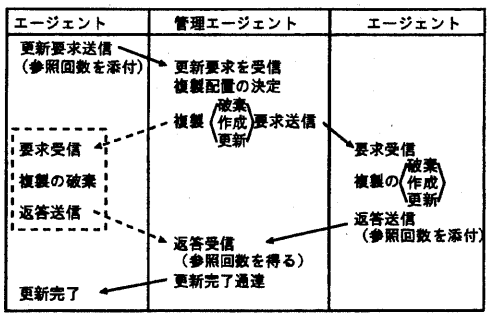

図 5: 更新時の処理

がプロセッサから参照された回数を知ることができる。ここで得た回数と前回までの回数との重み付け総和を基に次回の更新時に複製を所持させるエージェントを決定するわけである。各共有資源エージェントがプロセッサから参照された回数が複製の配置決定に反映されるのが1回遅れることになるが、本研究では各プロセッサにおける資源の参照頻度は急激に変化しないことを前提にしているので、このことは問題ないと考えている。

3.2 参照回数と複製配置について

複製を所持させるべきエージェントは参照頻度の高い順に選ばれるわけだが、いくつのエージェントに複製を所持させるべきかをこの節で説明する。

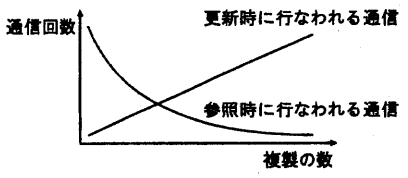

図 6: 複製の数と通信回数

本研究では、分散処理の効率を改善するために通信回数を削減しようと試みている。そこで、複製の配置と通信回数の関係を図6に示す。横軸は複製の数を示しており、縦軸はある更新が完了してから次の更新が完了するまでに行なわれる通信の回数を示

している。更新時には複製の数に比例した通信が行なわれるので、グラフは右上がりの直線になっている。一方参照時に行なわれる通信回数は、複製不所持エージェントによって参照が行なわれる時の通信回数の総和となるので、複製の数が増えるにつれて図のような曲線で減少していく。

全体の通信回数を最小にすることは、ある更新が完了してから次の更新が完了するまでに行なわれる通信の回数を最小にすることを意味している。従って、図6の2つのグラフの和のグラフが最小となるように複製の数を選べばいいことになる。これを図7に示す。

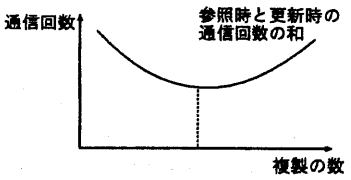

図 7: 最適な複製の数

4 実装方針

本研究では、共有資源の効率のよい実現の方法として、更新が行なわれる時に、各エージェントの参照回数を基に複製所持エージェントを決定する方法を提案した。前章において、分散問題解決の枠組を利用して複製の動的な配置の決定について述べた。この章では、提案した共有資源の実現方法の実装方法について述べる。

4.1 複製配置戦略部分の実装

本研究では、メタオブジェクトプロトコルを用いて協調処理をメタレベル計算として記述する枠組を利用して共有資源の実現を行なうことにした。そうすることにより、ユーザプログラムを書き換える必要なしに、共有資源の参照・更新時における処理をメタレベル計算として実現できる。そこでメタオブジェクトプロトコルを持つ Tiny-CLOS[Kic91] を使用して実装を行なった。Tiny-CLOS は Scheme 上のオブジェクト指向言語機構である。

Tiny-CLOS を使用して共有資源エージェントをオブジェクトとして実装する。共有資源エージェントのクラス（<shared-object>とする）は、管理エージェントのアドレス、複製の所持状態を示すフラグ、共有資源の値、参照回数、といったスロットを持っている。この中の共有資源の値を示すスロット body に対する参照・更新を再定義する方法を示す。

Tiny-CLOS ではオブジェクトのスロットを参照・更新する時に以下の関数を使用する。

(slot-ref *object slot-name*)

(slot-set! *object slot-name new-value*)

これらの関数は内部でオブジェクトのスロット毎に設定された参照子・代入子を使用してスロットの参照・更新を行なっている。つまり参照子・代入子を設定し直せば、通常と違った処理を行なうようになる。そこで、3章で述べた処理を行なう新たな参照子・代入子を作成しそれをスロット body への参照子・代入子に設定する。そうすると、共有資源オブジェクトのスロット body を slot-ref・slot-set!で操作する時に3章で述べた処理をメタレベルで行なわせることができる。

4.2 ユーザプログラムにおける実装

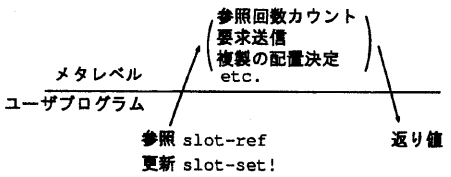

図 8: メタレベル処理

共有資源エージェントは、分散処理を行なうユーザプログラムにおいて使用される。そこで、ユーザプログラムにおいて本研究で実装したプログラムを使用する方法を述べる。まず、共有資源エージェント（<shared-object>のインスタンス）をスロットに持つオブジェクトを作る。そしてそのスロット（共有資源エージェント）への参照子・代入子をそのスロット（共有資源エージェント）のスロット body への参照・

代入とするように設定する。そうすると、ユーザプログラムでは、通常のスロットと同様に slot-ref・slot-set! で共有資源エージェントに参照・代入するだけで、3 章で述べた処理をメタレベルで行なわせることができる（図 8）。

5 実験・評価

本研究で提案した共有資源の実現方法は、各プロセッサにおける資源の参照頻度を基に複製の配置を動的に決定する方法である。提案方式の性能分析のために実験を行なった。実験内容は、4 つのクライアントが頻度を変化させて共有資源の参照を繰り返すテストプログラムを作成し通信回数を計測するものである。各クライアントの参照頻度の変化を図 9 に示す。読み取り複製方式と提案方式に対し実験を行なった。実験の結果を図 10 および図 11 に示す。

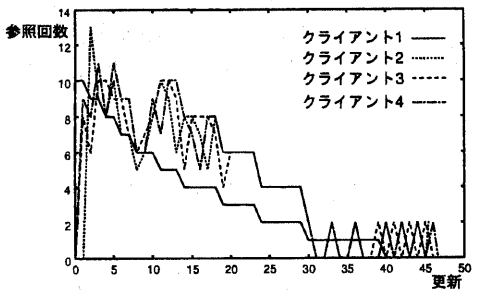

図 9: 参照頻度の変化

図 10 はテストプログラムの実行中に行なわれた通信回数を示し、図 11 は複製の数の推移を示している。参照回数が低いときの複製の数に違いが見られる。この時の更新時に行なわれる通信の回数が、両方式の全体の通信回数の差に影響を及ぼしているといえる。

読み取り複製方式	1150
提案方式	884

図 10: 両方式の通信回数

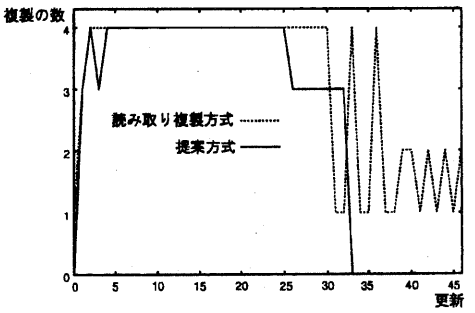

図 11: 両方式の複製の数の変化

6 おわりに

本研究では、分散環境下における従来の共有資源の実現方法では処理効率の改善が困難なことを示した。処理効率改善のために通信回数を削減することを考えた。効率のよい共有資源の実現のために分散問題解決を導入し、複製の配置決定のための戦略を提案した。また、すでに提案されている、協調処理をメタレベル計算として実現する方法を利用し、共有資源の複製の配置決定戦略を Tiny-CLOS により実装した。そして実験により読み取り複製方式と比較して通信回数の削減を確認できた。

今回は、研究の第一歩として 1 つの管理エージェントを設ける方法で共有資源の実現を行なった。その管理エージェントは固定されたものだったが、今後は管理エージェントの処理を分散化する方向で研究を進めていく。

参考文献

[Kic91] Gregor Kiczales. Tiny CLOS. Xerox, ftp://arisia.xerox.com/pub/openimplementations/, 1991.

[YNU95] 山崎賢治, 楢崎修二, 牛島和夫. メタレベル計算を用いた協調処理の実現. 情報処理学会研究報告 PRG, Vol. 95, No. 82, pp. 145–152, August 1995.