分散資源共有における動的な複製再配置

濱地, 弘樹
九州大学大学院システム情報科学府情報工学専攻 : 修士課程

楢崎, 修二
九州工業大学工学部

吉田, 紀彦
九州大学大学院システム情報科学府知能システム学専攻 :

下川, 俊彦
九州大学大学院システム情報科学府情報工学専攻 :

他

https://doi.org/10.15017/1498342

出版情報 : 九州大学大学院システム情報科学紀要. 3 (1), pp.75-80, 1997-12-22. 九州大学大学院システム情報科学研究院
バージョン :
権利関係 :

九州大学大学院
システム情報科学研究科報告
第3巻 第1号 平成10年3月

Research Reports on Information Science and
Electrical Engineering of Kyushu University
Vol.3, No.1, March 1998

分散資源共有における動的な複製再配置

濱地 弘樹* ・ 楢崎 修二** ・ 吉田 紀彦*** ・ 下川 俊彦† ・ 牛島 和夫†

Dynamic Copy Allocation in Distributed Resource Sharing

Hiroki HAMACHI , Shuji NARAZAKI , Norihiko YOSHIDA , Toshihiko SIMOKAWA
and Kazuo USHIJIMA

(Received December 22, 1997)

Abstract: 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 dynamically 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.

Keywords: Dynamic copy allocation , Distributed resource sharing , Distributed problem solving , Metalevel computing

1. はじめに

複数のプロセッサを用いて分散環境下で処理を行なうには，一般にプロセッサ間で資源の共有が行なわれる．ここで取り扱う資源とは，複製可能なデータのことである．共有資源の参照・更新を行なう時にはプロセッサ間で通信が必要となる．分散処理の効率を考える上で，通信コストは無視できない問題である．

共有資源の参照・更新時の通信回数を最少にする，効率のよい資源共有を行なうためには，各プロセッサに対して共有資源の複製を最適に配置することが必要となる．複製を所持させるべき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・更新頻度を知る必要がある．そこで本研究では，各プロセッサにエージェントを置き，分散問題解決の枠組を用いて共有資源の複製の動的な再配置を行なうことにする[1)2)3)]．

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[4)5)]．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中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を動的に再配置する分散資源共有方式を提案し，複製の再配置に必要な処理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. 従来の分散共有資源の実現方法

共有資源の参照・更新にはプロセッサ間で通信が必要となる．本研究では通信コストは計算コストに比べてはるかに大きいという前提で，資源共有の効率改善を行なう．そこで，資源共有の効率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使用頻度は急激に変化しないものと仮定している．本研究では強い一貫性の保持が必要な共有資源を取り扱うので，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，従来の分散共有資源の実現方法[6) 7)]と通信回数の観点から見たその特徴について述べる．

平成9年12月22日受付
* 情報工学専攻修士課程
** 九州工業大学 工学部
*** 知能システム学専攻
† 情報工学専攻

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方式は，**Fig.1**のように資源の唯一の複製を所持・管理する中央サーバを設置することである．中央サーバ方式では，資源不所持プロセッサが資源の参照・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．そのため資源不所持プロセッサが頻繁に資源を利用する場合は，通信回数が多くなり効率が悪くなる．また中央サーバに通信が集中すると中央サーバが隘路になる恐れがある．しかし資源の更新については，中央サーバの所持する一つの資源のみを更新すればよいので手間は少ない．

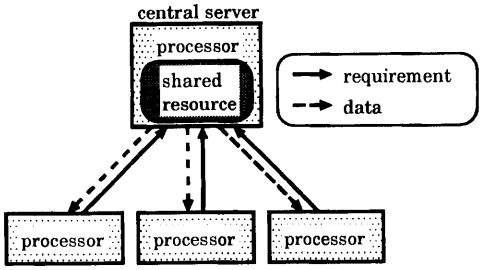

Fig.1　The central-server mechanism

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・更新を行なう際には常に中央サーバへの通信を必要とした．しかし，**Fig.2**のように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ようになり，参照時の通信を削減できる．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は多くなる．

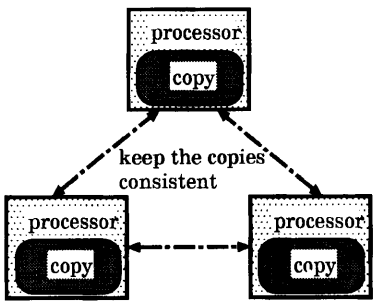

Fig.2　The full-replication mechanism

2.3 読み取り複製方式

分散共有メモリ[6) 7)]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際に，複製所持プロセッサに通信して資源の値を受けとる．その時に局所的に資源の複製を作成し，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際には，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．

この方式だと次の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，完全複製方式ほどではないにしても複製を所持するプロセッサ数が多くなる．そのため更新時に行なわれる通信回数は多くなるといえるが，新たな値を伝えるメッセージよりも複製破棄を伝えるメッセージの大きさは小さくて済む．

3. 複製配置の動的変更

2.章で述べた従来方式の特徴を総合して判断すると，プロセッサに資源の複製を所持させることによって資源の参照時の通信を削減できるが，複製を所持するプロセッサが増加すると更新時に複製間の一貫性を保持するための通信が増加する．よってむやみに複製を所持させるのではなく，最適な複製の配置を考える必要がある．複製の配置を決定する際には各プロセッサにおける資源の使用頻度を知る必要がある．また各プロセッサにおける資源の使用頻度は動的に変化する可能性もあるので一度決めた複製配置が常に最適とは限らない．

そこで本研究が提案する分散資源共有方式は，各プロセッサにおける共有資源の使用頻度を動的に検出し，その頻度情報を基に複製を動的に再配置する，という方式である．この方式を，分散問題解決の枠組を利用して実現する．

3.1 分散問題解決

分散問題解決とは，複数の，エージェントと呼ばれる独立した処理主体が協調を行ない1つの問題を効率よく解いていく問題解決法である．分散問題解決で問題を解いていく上でその実行環境は動的に変化する．よって問題解決に向けたエージェントの動作を，その時点での実行環境に応じて決定した方が効率よく問題を解決できる．しかし各エージェントは実行環境の部分情報しか持たないため，1つのエージェントだけでは効率のよい動作決定を行なうことはできない．効率のよい動作決定を行なうためには，エージェントは実行環境の状況を知る必要がある．そのために他のエージェントの持つ情報や状態を通信や推論で得て，その情報を基に実行環境のモデルを

作成する．そうやって作成した実行環境のモデルを用いて全体の処理効率を上げるように各自の動作を動的に決定する．このような処理を協調と呼ぶ．

3.2 分散問題解決による分散資源共有

ここで，分散問題解決の枠組を利用して共有資源の複製の配置を動的に変更する分散資源共有方式について述べる．まず，各プロセッサにそれぞれ1つのエージェントを置く．それらのエージェントは自分の置かれているプロセッサにおける共有資源の使用頻度を調べる．そしてその頻度情報をやりとりして最適な複製の配置を決定する．

エージェントの持つ部分情報とは自分で調べたプロセッサにおける共有資源の使用頻度であり，実行環境のモデルとは全プロセッサにおける資源の使用頻度である．

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．プロセッサは共有資源を参照・更新したい時には自分に置かれている共有資源エージェントにアクセスする．プロセッサからアクセスされた時に，エージェントは資源の参照回数を数えたり複製配置の決定などの処理を行なう．また共有資源の複製はエージェントが管理する．プロセッサからアクセスを受けてエージェントが動作を開始する，という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．実装の詳細は4.章で述べる．複数の異なる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．

3.3 管理エージェントを置いた方法

分散問題解決の枠組を利用した分散資源共有の手始めとして，1つの特別なエージェントを置いた方法を考えた．それを**Fig.3**に示す．ここで特別なエージェント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を決定する権限を持っている．

複製の再配置方法を簡単に説明すると，資源の更新処理の間を1サイクルと考えて，各プロセッサにおける1サイクル間での共有資源の参照回数を各共有資源エージェントが調べる．その参照回数を管理エージェントに集めて，その情報を基に管理エージェントが複製の最適配置を決定する．

次に共有資源エージェントがプロセッサからアクセスされた時の処理を詳しく説明する．

資源の参照時の処理を**Fig.4**に示す．共有資源エージェントはプロセッサから参照アクセスされた時に，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，すなわち要求元のプロセッサにおける資源の参照回数を計数しておき，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．その際，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．

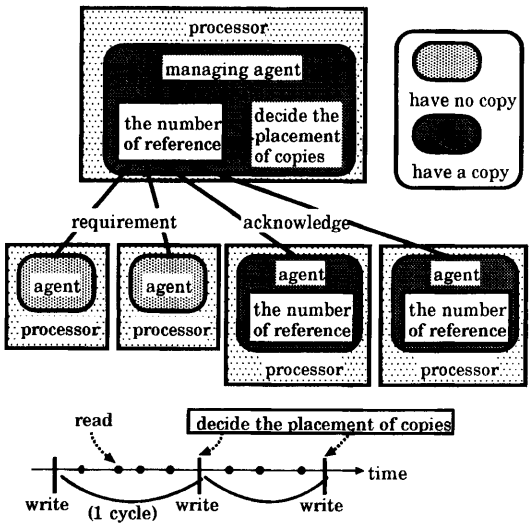

Fig.3 Dynamic copy allocation with a managing agent

また，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．2回とは，参照要求の送信とそれに対する返答である．一方，複製所持エージェントは参照時に通信を必要としない．

requesting agent	managing agent
If agent have no copy count up the reference Else send data request	
	Receive request Count up the reference Send response
recieve response (don't make a new copy)	
Read data	

Fig.4 The read operation

次に，資源の更新時の処理を**Fig.5**に示す．共有資源エージェントは，プロセッサから更新アクセスされた時

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，各プロセッサにおける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．

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が必要となる．それも加えると，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．

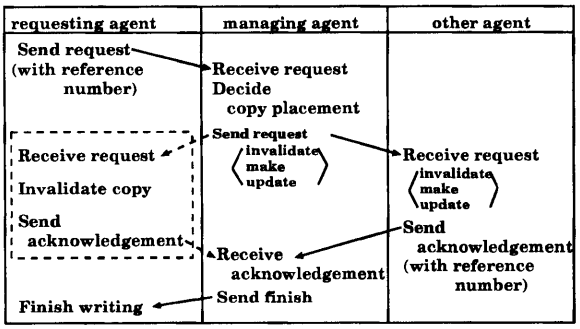

Fig.5　The write operation

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参照回数とを足して2で割る重み付け総和を基に次のサイクル間での見込みの参照回数を算出する．資源の更新が行なわれる度に，見込みの参照回数を基にして，最適な複製配置を決定し複製の再配置を行なう．

各プロセッサにおける参照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．

3.4　参照回数と複製配置について

ここで，見込みの参照回数を基に複製の配置を決定する方法を述べる．複製を所持させるべきエージェントは参照回数の多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，すなわち最適な複製の数は次のようにして決定する．

複製の数と1サイクル間での通信回数の関係を**Fig.6**に示す．横軸は複製の数を示しており，縦軸は1サイクル間に行なわれる通信回数を示している．1回の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**Fig.6**のような曲線で減少していく．

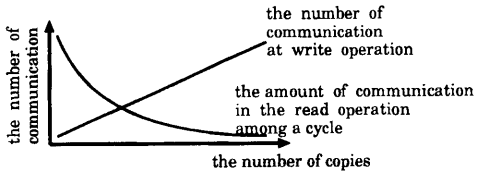

Fig.6　The relation between the number of copies and the number of communication

全体の通信回数を最小にすることは，1サイクル間の通信回数を最小にすることを意味している．従って，**Fig.6**の2つのグラフの和，すなわち**Fig.7**に示すグラフが最小となる点から最適な複製の数が決定される．すなわち参照回数の多い順に**Fig.7**で求めた数のエージェントに複製を所持させた状態が最適な複製配置ということになる．

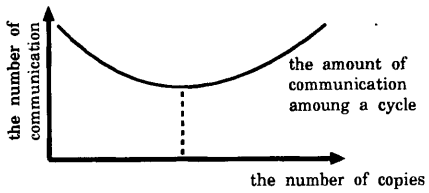

Fig.7　The most suitable number of copies

ここで述べた処理を共有資源の更新処理の度に行ない，複製配置を動的に最適化する．

4. 実　　装

本研究では，共有資源の効率のよい実現の方法として，更新が行なわれる度に，各プロセッサにおける資源の参照回数を基に複製を再配置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，提案した資源共有方式の実装方法について述べる．

4.1 複製の再配置処理の実装

本研究では，メタオブジェクトプロトコル[8]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[5]を利用して提案した分散資源共有方式の実装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・更新時におけるエージェントの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[9]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定義する．共有資源エージェントのクラスを<shared-object>とした．<shared-object>のインスタンスは，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，局所的に保持する複製の値，参照回数，といったスロットを持っている．共有資源の値を示すスロットをbodyとして，スロットbodyに対する参照・代入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・代入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.章で述べた複製の再配置のための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，複製の再配置に必要な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

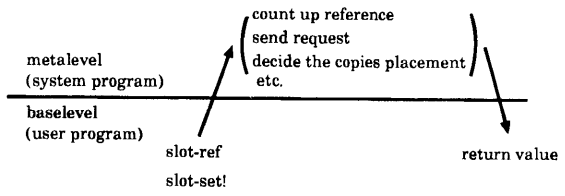

Fig.8 The metalevel process

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，すなわち<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への参照子・代入子をそのスロットが持つスロットbody への参照・代入を行なうように設定する．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を参照・代入するだけで，複製の再配置に必要な処理をメタレベルで行なわせることができる．これを**Fig.8**に示す．

4.3 分散環境での実装

<shared-object>のスロットbodyに対する参照子・代入子は，3.章で述べた複製の再配置のための処理を行なうように記述している．この中では管理エージェントとの通信についても記述している．エージェント間通信はTCP/IPのソケットを利用して実装している．Schemeの処理系として使用したGuile[10]ではUnixのシステムコールが利用できるため，ソケットを使用して通信を行なうプログラムを扱うことができる．

5. 実 験・評 価

本研究で提案した共有資源の実現方法は，1サイクル間での各プロセッサにおける資源の見込みの参照回数を基に複製の配置を動的に変更する方法である．提案方式の性能分析のために読み取り複製方式との比較実験を行なった．

実験内容は，4つのプロセッサがサイクル毎に頻度を変化させて共有資源の参照を行なう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．**Fig.9**に示すように，各プロセッサの参照頻度を徐々に減少させていき，通信回数を計測した．実験の結果を**Fig.10**および**Fig.11**に示す．

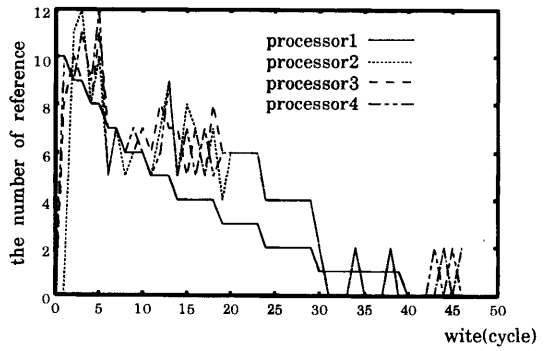

Fig.9 The change of reference frequency. (sample data)

Fig.10は複製の数の推移を示し，**Fig.11**は通信回数の累積を示している．

10回目から30回目くらいまでの更新の間は，**Fig.10**より両方式とも4つのプロセッサすべてが複製を所持している．このとき**Fig.11**のグラフの傾きに差がある．これは

読み取り複製方式では複製の破棄後の最初の参照時には通信を必要とするからである.

また, **Fig.10**より, 提案方式では各プロセッサにおける参照回数が少ないと複製を所持させていない. **Fig.11**のグラフの傾きを見ると読み取り複製方式よりも傾きが低くなっている. これらより, 提案方式では参照回数が少ない時には複製を所持させないと判断したため通信回数を抑えることができているといえる.

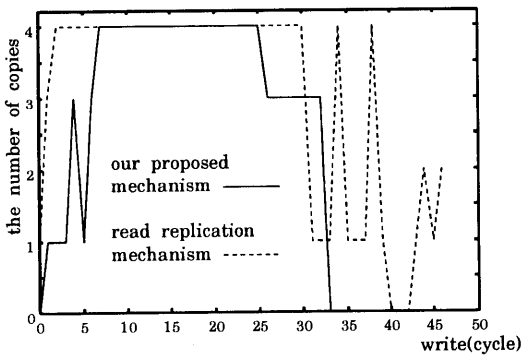

Fig.10 The change of number of copies

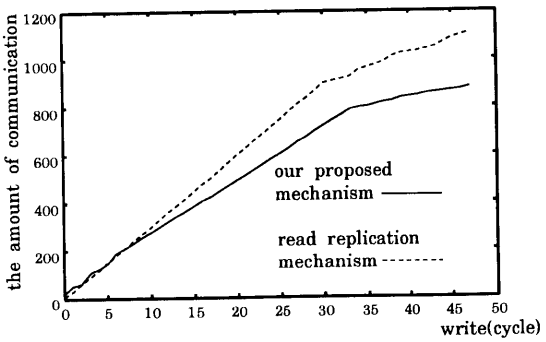

Fig.11 The total number of communication

上記の実験は単に参照と更新を行なうだけのテストプログラムによるものだが, 具体的な適用例として巡回セールスマン問題を取り上げた. 処理内容は, 複数のエージェントがそれぞれ最短経路を探索して, それまでに見つかっている最短経路の距離をエージェント間で共有するというものである. この場合その時点での最短経路の距離に対する参照・更新頻度がほとんど変化しなかったため, 提案方式でも複製の配置はほとんど変動しなかった. だが, 最適な複製配置を保つことはできていた.

6. おわりに

本研究では, 従来の分散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. そこで分散問題解決を導入し, 複製の配置を動的に最適化する方式を提案した. 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, Tiny-CLOSを用いて, 複製の再配置に必要な処理を資源の参照・更新に付随するメタレベル計算として実装した. 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, すなわち効率の改善を確認した.

今回は通信回数だけで資源共有の効率を測っていたが, 実際はそれ以外にも様々な要素が絡んでくる. 今後は他の要素も考慮して複製の動的再配置を考えていく. また現在は更新処理の度に複製の再配置を行なっているが, 実際には複製配置は毎回変わるものではないので, 配置変更をいつ試みるかについて考察していく. そして, 研究の第一歩として1つの管理エージェントを置いた方法を考えたが, 複製の数が増えると更新時に管理エージェントに負荷が集中する. 管理エージェントの負荷を分散するために, 管理エージェントの数を増加させる方法が考えられる. しかし管理エージェントの数が増加すると管理エージェント間の一貫性保持の手間が大きくなる. そこで最適な管理エージェントの数を動的に決定する方法が今後の課題となる.

参考文献

1) 濱地 弘樹. 分散問題解決による共有資源の実現. 九州大学卒業論文,February 1996.
2) 濱地 弘樹 , 楢崎 修二 , 吉田 紀彦 , 牛島 和夫. 分散問題解決による共有資源の実現. 情報処理学会「プログラミング」研究会報告 97-PRO-14, pp.31-36
3) 濱地 弘樹 , 楢崎 修二 , 吉田 紀彦 , 牛島 和夫. 分散問題解決による共有資源の効率的な実現. 情報処理学会第 55 回全国大会 3T-2 (1997)
4) Shuji Narazaki, Hiroomi Yamamura and Norihiko Yoshida, "Strategies for Selecting Communication Structures in Cooperative Search", Int'l Journal of Cooperative Information Systems, 4:4, 405-422 (December, 1995)
5) 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.
6) Michael Stumm and Songniam Zhou. Algorithms Implementing Distributed Shared Memory. *IEEE Computer*, Vol. 23, No. 5, pp. 54–64, May 1990.
7) Jelica Protic, Milo Tomasevic and Veliko Milutinovic, Distributed Shared Memory; Concepts and Systems, IEEE Computer Society (1998)
8) Gregor Kiczales, Jim de Rivieres and Daniel G. Bobrow, The Art of the Metaobject Protocol, MIT Press (1991)
9) Gregor Kiczales. Tiny CLOS. Xerox, ftp://arisia.xerox.com/pub/openimplementations/, 1991.
10) Guile iii. ftp://ftp.cygnus.com/pub/guile/, 1993.