プログラミング－言語・基礎・実践－ 18－19
（１９９４．７．２２）

分散 Linda システムの構築

茅野 剛史　　多田 好克
電気通信大学大学院 情報システム学研究科

協調言語である Linda は、現在 UNIX ワークステーション上で実現されたものが多い。また、現行の Linda システムではデータ通信の手段であるタプル空間がひとつのワークステーション上にしかないものが多く、そのためにさまざまな問題が発生している。

本研究では、タプル空間がひとつしかないことによって起こる問題を解決するために、タプル空間が複数の計算機上に分散して存在する Linda システムを提案する。また、効率をあげるためにアプリケーションのデータの扱いによって、分散のさせ方を変更するアプリケーション指向のアルゴリズムを考案する。今回の実現では、既存のシステムを改良し、Linda のセマンティクスを用いて分散システムを記述した。

An implementation of distributed Linda system

Takeshi Kayano, Yoshikatsu Tada

The University of Electro-Communications
Guraduate School of Information Systems,
1-5-1 Choufugaoka,Choufu-shi Tokyo 182 JAPAN

Today, coordination language Linda is implemented on UNIX workstations. But, Linda now in use has poor performance problems because of its single Tuple Space which is the only way to communicate.

In this paper, we propose new Linda system that has distributed Tuple Space to solve problems arise from the single Tuple Space of usual Linda systems. And, we contrive distributed Tuple Space algorithm, which changes several implementations according to a behavior of applications. We implemented this system using Linda semantics themselves.

1 はじめに

並列プログラミングには、処理言語(computing language) と協調言語 (coordination language) が必要である。このふたつの言語によって、並列プログラミング言語を作る事が出来る[2]。

つまり、以下のように表す事が出来る。

処理言語 + 協調言語 = 並列言語

処理言語 計算やローカルなデータオブジェクトの処理に使用する (C や FORTRAN などの従来の言語)。

協調言語 並列実行や通信などに使用する (Linda など[1])。

処理言語と協調言語の機能をひとつの並列言語 (例えば Occam など) に含めることもできるが、新たにその言語を覚えるのは大変である。しかし、処理言語と協調言語とをあわせて作った並列言語は、従来の言語に並列化のためのいくつかの演算子が加えられたものなので、新たに言語を覚えなおす必要がないため便利である。

現在、協調言語である Linda のシステムが UNIX ワークステーション上で実現されている[5][6]。本研究ではこのシステムの問題点を考え、それを解決するための方法を考える。

本稿では、第 2 章で、まず Linda と Linda の演算子について説明する。第 3 章では、現行の Linda システムについて説明し、問題点について述べる。第 4 章では、その問題点を解決するためのシステムの構築法について議論する。第 5 章では、構築したそれぞれのシステムの評価を行う。第 6 章では、今後の課題について述べる。

2 Linda

Linda[2] は、Yale 大学の Carriero、Gelernter らによって考案された協調言語である。処理言語である C や FORTRAN とあわせる事によって、C-Linda や FORTRAN-Linda などの並列言語を作る事が出来る。

Linda システムでは、データの受渡しは "タプル" というデータ構造体によって行われる。ユーザは以下にあげる演算子を使用し、タプル空間という一種の論理的な共有メモリを介してタプルのやりとりを行う。

2.1 Linda の演算子

Linda で使用する演算子を以下に説明する。

out 演算子: out 演算子は、タプル空間へタプルをストアする。

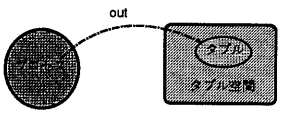

図 1: out 演算子

in 演算子: in 演算子は、タプル空間から引数とマッチするタプルを取り出し、そのタプルをタプル空間から削除する。マッチするタプルがない場合、マッチするタプルがタプル空間に生じるまでプロセスをブロックする。マッチするタプルが生じた場合、そのタプルを返し復帰する。

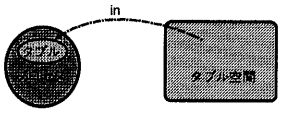

図 2: in 演算子

read 演算子: read 演算子は、ほぼ in 演算子と同様の働きをする演算子であるが、in 演算子と異なりタプル空間内のタプルを削除しない。

[1] Linda は Scientific Computing Asociates 社の登録商標である。

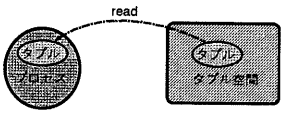

図 3: read 演算子

inp 演算子と readp 演算子: inp 演算子と readp 演算子は、それぞれ in 演算子と read 演算子と同様な動作をする演算子である。しかし、これらはマッチするタプルがタプル空間内に存在しない場合、プロセスをブロックせずにすぐ復帰する。

3 従来の Linda システム

従来の Linda システムでは、ネットワーク中の計算機のひとつにだけタプル空間が存在し、プロセスはその計算機と通信しタプル空間へのアクセスを行う。

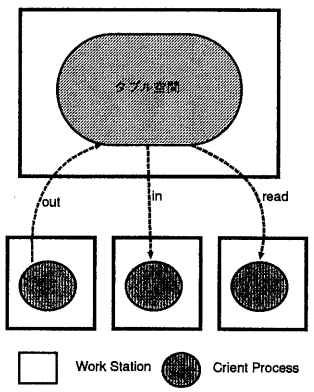

図 4: 従来の Linda システムの概念図

しかし、このようなシステムには、タプル空間のある計算機に負荷がかかったり、パケットが集中するといったような問題点が存在する。

これらの問題点の解決法のひとつとして、タプル空間を分散させるという方法が考えられる。

4 分散 Linda システム

分散 Linda システムでは、従来の Linda システムと異なり、タプル空間がネットワーク内の各計算機ごとに存在する。タプル空間を分散させる事によって、従来のシステムの問題点は解決できると考えられる。

4.1 改良の方法

本研究では、TS-system[5] を改良して分散 Linda システムを実現する。改良の方法としては、サーバを改良する方法とクライアントを改良する方法がある。

4.1.1 サーバに手を加える

サーバ部分に手を加え、クライアント部分には手を加えない。この方法では、ユーザはタプル空間がどのような方法で実現されているかを意識する必要がなく、従来の Linda システムのプログラムをそのまま使用する事が出来る。

サーバは、それぞれのアルゴリズムに従い、クライアントの実行した演算子に対して処理を行う。

4.1.2 クライアントに手を加える

クライアント部分に手を加え、サーバ部分には手を加えない。この方法では、ユーザはタプル空間がどのような方法で実現されているかを意識する必要があり、ユーザプログラムにあったアルゴリズムを使用しなくてはならない。もし、そのプログラムに合わないアルゴリズムを使用すると、効率が悪くなる可能性がある。

今回の実現では、各アルゴリズムの検証も兼ねているため、クライアントのみの改造によって分散 Linda を実現する事にする。また、アル

ゴリズムの記述にはLindaのセマンティクスを使用する。

4.2 具体的な実現法

以下に、タプル空間の分散のための各アルゴリズムを示す。

4.2.1 out-Broadcast型システム

この方法では、全ての計算機上のタプル空間は同一のタプルを持つことになる。

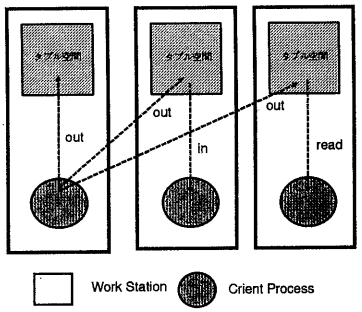

図5: out-Broadcast型システム

out 演算子 タプル空間を持っている計算機のリストをreadし、リストに入っている計算機のタプル空間へタプルをoutする。タプルの第一要素には、誰がoutしたタプルかわかるようにoutした計算機名をつけておく。

in 演算子 マッチするタプルをローカルにreadし、タプルが返ってきたらそのタプルをoutした計算機からをinpする。成功したら他の計算機から削除してまわり、その後returnする。失敗した場合は、そのタプルは他の計算機にinされたということなので、最初(readのところ)までもどって繰り返す。

inp 演算子 マッチするタプルをローカルにreadpし、マッチしなかったらreturnする。マッチした場合は、そのタプルをoutした計算機からをinpする。成功したら他の計算機から削除してまわり、その後returnする。失敗した場合は、そのタプルは他の計算機にinされたということなのでreturnする。

read 演算子 ローカルにreadする。

readp 演算子 ローカルにreadpする。

out-Broadcast型システムを使用した場合、read, readpはローカルにタプル空間に接続し検索を行うため早いが、out, in, inpはブロードキャストを行うため遅くなる。

よって、この方法はreadを多く繰り返すアプリケーションに最適であると思われる。

4.2.2 in-Broadcast型システム

この方法では、各計算機上のタプル空間は自分がoutしたタプルのみを保持することになる。

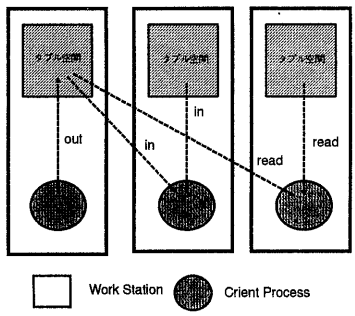

図6: in-Broadcast型システム

out 演算子 ローカルにタプルをoutする。

in 演算子 ローカルにinpを行ない、マッチしなかった場合はタプル空間をもつ計算機のリストをreadし、順番にinpしていく。マッチした時点でタプルを返す。全部inpしてみてマッチしなかった場合は、しばらくsleepしてからまた最初にもどって繰り返す。

inp 演算子 ほぼin 演算子と同様だが、全部inp してみてマッチしなかった場合はsleep せずにreturn する。

read 演算子 ローカルにreadp を行ない、マッチしなかった場合はタプル空間をもつ計算機のリストをread し、順番にreadp していく。マッチした時点でタプルを返す。全部readp してみてマッチしなかった場合は、しばらくsleep してからまた最初にもどって繰り返す。

readp 演算子 ほぼread 演算子と同様だが、全部readp してみてマッチしなかった場合はsleep せずにreturn する。

in-Broadcast型システムを使用した場合、out はローカルにタプル空間に接続しタプルをストアするため速いが、in, inp, read, readp はブロードキャストを行うため遅くなる (ただし、ローカルにマッチした場合は速い)。

また、各計算機のタプル空間を合計したものが、従来のLinda におけるタプル空間と同じ容量になるので、計算機ひとつあたりのメモリ使用量は少ない。

よって、この方法はout を多く繰り返すアプリケーションに最適であると思われる。

4.2.3 hash 型システム

この方法では、各計算機上のタプル空間はhash 関数によりhash されたタプルを持つことになる。in, inp, read, readp 演算子の時、タプルの要素が "." の場合、その要素には任意の文字列がマッチする。今回はタプルの第一要素によりhash を行うので、第一要素が "." の時はhash ができない。

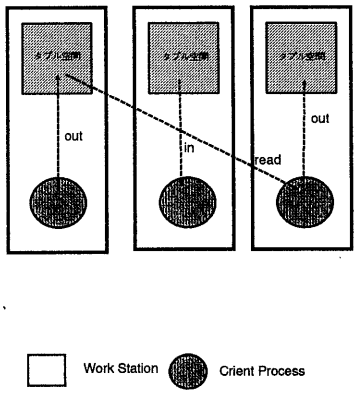

図 7: hash 型システム

第一要素が "." 以外の時 タプルの第一要素でhash して対象となる計算機を決め、その計算機のタプル空間に対して操作を行う。

第一要素が "." の時 hash によって検索することがができないので、in-Broadcast 型のように各タプル空間からマッチするタプルを検索するしかない。

hash 型システムを使用した場合、それぞれの演算子はhash 関数によって接続するタプル空間を得る。そのため、計算機間のパケットが少なくて済む。また、このシステムもin-Broadcast 型と同様に計算機ひとつあたりのメモリ使用量が少なくなる。

よって、この方法はout-Broadcat 型システム、in-Broadcast 型システムでは効率化できないようなアプリケーションに最適であると思われる。

5 システムの評価

本章では、従来のLindaシステムと分散Lindaシステムの効率を比較してみる。

5.1 計算機のロード

5.1.1 read が多い場合

クライアントは、out 1回に対してread 10,000回を行った。

従来のLindaシステムを使用した場合、各計算機のロードは図8のようになる。タプル空間のある計算機のロードが、他のクライアントしかない計算機のロードに比べて高い事がわかる。

out-Broadcat型、in-Broadcast型システムを使用した場合、各計算機のロードは図9、図10にのようになる。各計算機にタプル空間が存在するため、ロードが平均化しているのがわかる。また、各計算機の平均ロードが従来のLindaより高くなるが、実行時間自体は短くなっている。in-Broadcast型は、この場合ローカルにタプルがマッチしているため速くなっている。

hash型システムを使用した場合、各計算機のロードは図11にのようになる。実験に使用したデータにより、outされるタプル空間がかたよるため、特定の計算機のロードが高くなっている。

5.1.2 out が多い場合

クライアントはout 10,000回に対して、read 1回を行った。

従来のLindaシステムを使用した場合、各計算機のロードは図12のようになる。readが多い場合とほぼ同様になるが、outはreadより処理が軽いため実行時間は短くなっている。

out-Broadcast型システムを使用した場合、各計算機のロードは図13のようになる。outが各タプル空間に、タプルをストアして行くので時間がかかってしまっている。

in-Broadcast型システムを使用した場合、各計算機のロードは図14のようになる。outがローカルに行われるので、実行時間も速く、ロードも低い。

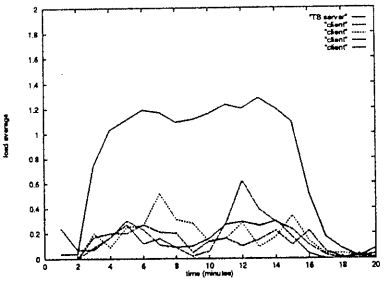

図 8: read が多い時の従来の Linda

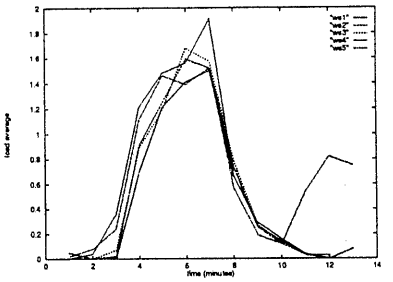

図 9: read が多い時の out-Broadcast 型

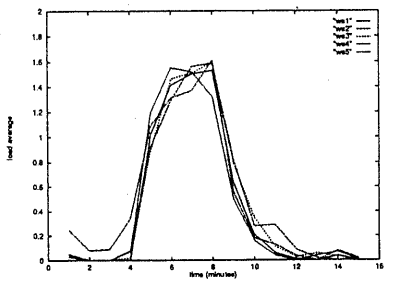

図 10: read が多い時の in-Broadcast 型

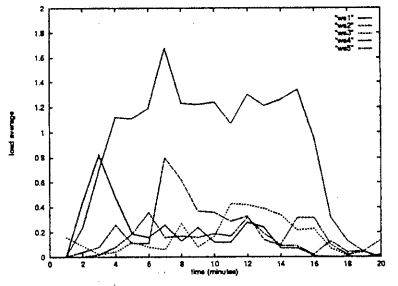

図 11: read が多い時の hash 型

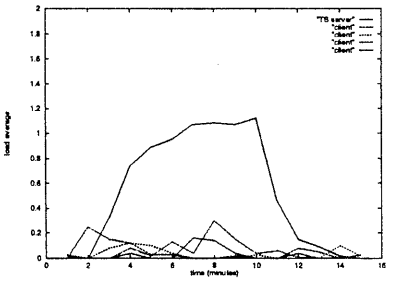

図 12: out が多い時の従来の Linda

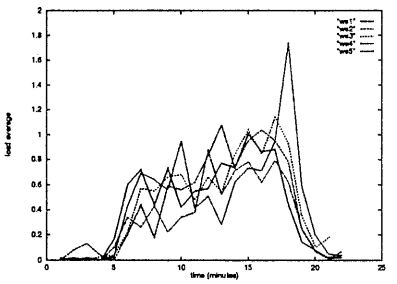

図 13: out が多い時の out-Broadcast 型

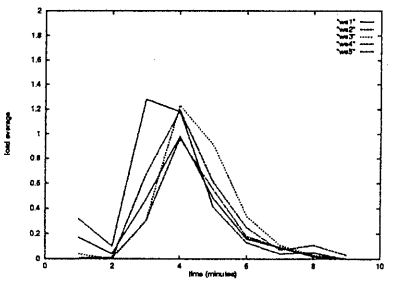

図 14: out が多い時の in-Broadcast 型

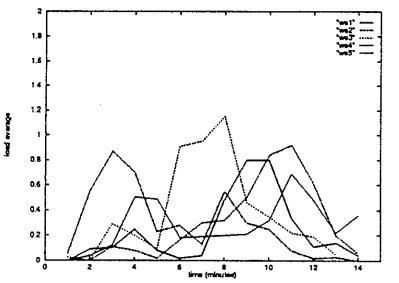

図 15: out が多い時の hash 型

hash 型システムを使用した場合、各計算機のロードは図 15のようになる。特定のタプル空間へ out が集中するため、順番にロードが高くなっている。

5.2 実行時間

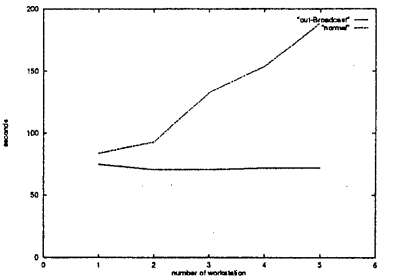

図 16: read が多い時の実行時間

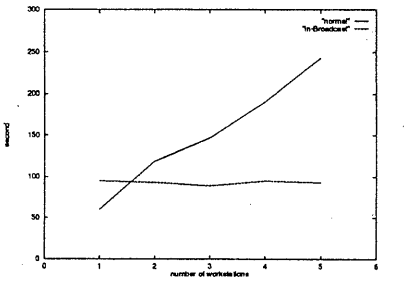

図 17: out が多い時の実行時間

図 16は、従来の Linda システムと out-Broadcast 型システムを使用し、クライアントの個数を増やしていった場合の実行時間を示した。各クライアントは out 10,000 回に対して、read 1 回を行った。read を行う回数が多い場合、クライアントの数が増えても out-Broadcast 型システムはほとんど実行時間がかわらないことがわかる。

図 17は、従来の Linda システムと in-Broadcast 型システムを使用し、クライアントの個数を増やしていった場合の実行時間を示した。各クライアントは out 1 回に対して、read 10,000 回を行った。out を行う回数が多い場合、

クライアントの数が増えてもin-Broadcast型システムはほとんど実行時間がかわらないことがわかる。

5.3 評価

out-Broadcast型システムを使用した場合

readを行う回数が多い場合、クライアントの数が増えても実行時間はほとんどかわらなかった。また、従来のLindaと比べてロードが平均化していることがわかった。

in-Broadcast型システムを使用した場合

outを行う回数が多い場合、クライアントの数が増えても実行時間はほとんどかわらなかった。また、従来のLindaと比べてロードが平均化していることがわかった。

hash型システムを使用した場合

hashの仕方や、データの内容により、各計算機のロードにかたよりが出る事がわかった。

6 今後の課題

分散Lindaシステムには、以下のような問題がまだ残されている。

- Lindaのセマンティクスのみでの実現のため無駄が多い。
- アルゴリズムをあらかじめユーザがきちんと理解していない場合、余計遅くなる可能性がある。
- 効率をはかる時に、演算子のみの(内容のない)プログラムを使用したため、現実にそぐわない可能性がある。

これらの問題を解決するための方法として、以下のようなことが挙げられる。

- クライアント側だけでなく、サーバにも手を加え効率化をはかる。
- アプリケーションによって自動的に、アルゴリズムを変更するようなシステムを考える。
- 実際に何かの処理をするプログラムを書き、それを実行して効率をはかる。

今後はこの方針にしたがって研究を進める。

参考文献

[1] N. Carriero and D. Gelernter, 村岡 洋一訳, 並列プログラムの作り方, 共立出版, 東京 (1993).

[2] Carriero, N. and Gelernter, D., Coordination Languages and their Significance, Comm. of ACM, Vol.35, No.2, pp.97-107, Feb. 1992.

[3] 立山 義祐, タプル空間通信を使ったユーザインタフェースの構成法, 東京大学大学院工学系研究科修士論文, Mar. 1994.

[4] N. Carriero and D. Gelernter, S/Net's Linda Kernel, ACM Trans. of Computer Systems, Vol. 4, No. 2, pp.110-129, May 1986.

[5] 寺田 実, TS-system について, TS-system Manual, Sep. 1992.

[6] 増井 俊之, Perl による Linda サーバの設計と実装, Dec. 1993.