計算機アーキテクチャ 142−25
ハイパフォーマンス
コンピューティング 85−25
（2001. 3. 9）

PCクラスタを用いた分散処理システムClopにおける動的負荷分散方式の実装と評価

近藤 敦 † 大鎌 広 ‡ 藤原 祥隆 †
† 北見工業大学 ‡ 室蘭工業大学

大規模計算をPCクラスタを用いて分散処理し処理時間の短縮を効果的に行うためには、各PCで実行する処理のCPU占有時間を均等に調整する負荷分散が必要がある。本稿ではClop用の動的負荷分散方式を設計、実装している。分散処理システムClopはシングルクライアント-マルチサーバ構成のPCクラスタ用の、筆者らが提案するメッセージプーリング方式のプログラミングライブラリである。設計した動的負荷分散システムは各サーバの状態を中央監視部に集める集中管理型を採用した。実装した動的負荷分散方式の性能をMandelbrot集合作成によって評価した。

Implementation of dynamic load balancing system for distributed processing system Clop on PC cluster

Atsushi KONDOU † Hiroshi OHKAMA ‡ Yoshitaka FUJIWARA †
† Kitami Institute of Technology
‡ Muroran Institute of Technology

In this paper, dynamic load balancing system for Clop is designed. The distributed processing system Clop is a programming library on PC cluster of single-client multi-server based on Message-Pooling method. Dynamic load balancing system is collecting each server status with a central job dispatcher. Performance of the method is investigated with computing Mandelbrot set.

1 はじめに

PCクラスタを用いた分散処理システムClopにおける動的負荷分散方式の設計、実装、評価を行う。

ネットワークに接続した複数のPC/WSを協調動作させ処理時間の短縮を行うためには、各PC/WSで実行する計算処理に必要な通信処理を隠蔽することが有効であるが、通信の隠蔽を明示的に記述することはプログラマにとって負担になっている。

そこで筆者らは、プログラマにPC間のデータ移動を意識させず、通信処理時間と計算処理時間の重ね合わせを自動で行う**メッセージプーリング方式** [1,2,3] を提案し、これを実装したシングルクライアント-マルチサーバ型PCクラスタ用の分散処理システム **Clop**(**C**luster **o**f message **p**ooling) を開発している。

システムの性能低下を抑えるため負荷の均衡を保つには、負荷分散を考慮した上で各処理単位へ処理を依頼していく必要がある [4,5]。しかしClopでは処理を依頼するPCを決定するのはユーザであり、あらゆる問題に対して常に最適な負荷分散を行うことは容易で

はない。

そこで本稿では Clop における動的負荷分散方式を提案、実装し Mandelbrot 集合作成による性能評価によって有効性を示した。

2 Clop の概要

Clop では全計算機をユーザへのインターフェースのための 1 台の **Clop クライアント**と、Clop クライアントからの命令を受けつけ実行する **Clop サーバ**の 2 つに分けている。さらに全ての Clop サーバにはユニークな**サーバ ID** がつけられる。

2.1 Clop クライアント

Clop クライアントはユーザプログラムを実行し、命令やデータを Clop サーバに送信する。ネットワークを介した PC 上のデータの流れは把握しにくく、PC 間のデータ移動処理の記述はユーザにとって負担である。これを軽減するため Clop ではユーザインターフェースとして**リモートポインタ**クラスが提供され、これによって Clop サーバ上のデータを間接的に操作する。

リモートポインタはデータ自体の格納領域ではなくデータ ID とデータが存在するサーバ ID のリストをメンバとして持っており、プログラマは Clop サーバ上のデータを格納場所を意識せずに統一的に扱うことができる。

また Clop サーバで処理を実行させるには、まず Clop が提供する基底クラスから派生させた**ユーザ定義関数**を作成し、そのインスタンスを Clop クライアントに渡すことで行う。

2.2 Clop サーバ

Clop サーバではメッセージプーリング方式の実装によって、Clop クライアントからの実行命令を受信順序ではなくデータの依存性をもとに実行していく (図 1)。命令の実行順序が可変であるためデータの受信待ちによるブロック時間を軽減できる。

さらに Clop サーバには通信、計算処理を実行するスレッドが各々実装されている。これによって通信処理と計算処理を自動で重ね合わせ、通信時間の隠蔽を行う。

3 Clop における動的負荷分散方式の必要性

PC クラスタでは、システム全体の処理時間は処理時間の最も大きい PC に依存する。各 PC 間の負荷のばらつきはピーク性能の低下を引き起こすのでこれを避けるためには効果的な負荷分散が必要である。

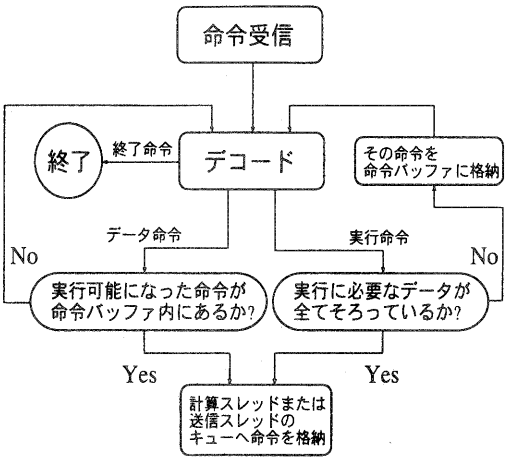

図 1: メッセージプーリングアルゴリズム

これまで Clop ではユーザが明示的に負荷分散を記述する必要があった。

このような、問題に応じてあらかじめ負荷分散を記述しておく静的負荷分散方式では、ユーザがあらかじめ各 PC で計算させる全ての問題の負荷について既知であることが必要である。

各サーバ上で行う処理の負荷を考慮した上で計算機の処理実行時間のばらつきを抑えるためには、リアルタイムに負荷分散を行う**動的負荷分散**が必要である。

そのため本稿では Clop に、ユーザが明示的に負荷分散を記述せずに負荷分散を行える動的負荷分散システムを実装する。

3.1 Clop における動的負荷分散方式の検討

動的負荷分散を実装する方法としては、

- システムに唯一つの負荷分散スケジューラが全計算機の負荷分散を一手に引き受ける集中制御方式
- 計算機同士が互いの負荷を調整しながら自律的に負荷分散を行う分散制御方式

などが考えられる。

Clop はシングルクライアント-マルチサーバ型であり実際に処理を実行するのはサーバであるため、サーバでは処理実行以外の負荷を最小に留める必要がある。そのためクライアントに負荷分散を行うスケジューラ

を用意し、サーバの負荷の均衡を保ちながら処理実行命令を発行する中央制御方式を選んだ。

さらに本稿では各サーバの計算時間のみを考慮した動的負荷分散アルゴリズムを用いた。

3.2 実装方法

クライアント上の負荷分散スケジューラがサーバの状態を常に把握しながら命令を発行していく。

クライアントからの命令送信がサーバの状況によって行われるので実際に送信されるタイミングはユーザからの要求とは異なる。そのためユーザが発行した関数オブジェクトを一時クライアント内にバッファリングし、スケジューラからの指示で送信専用スレッドが各サーバに送信する。

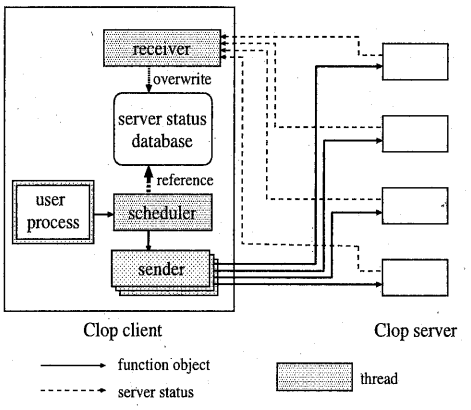

図 2: 集中管理による負荷分散方式の提案システム構成

サーバの計算スレッドは計算終了時にクライアントに自分が現在持っている処理実行命令数、データ個数などの**リソース**を即座に送信する。そして受け取ったクライアントのサーバ監視スレッドによって**サーバ状態データベース**が更新される (図 2)。

クライアントのスケジューリングスレッドはサーバ状態データベースを参照し、格納している命令数が 1 になっているサーバが無ければスリープする。サーバ状態データベースの更新が伝えられるとサーバ状態データベースを参照し、最も命令数の少ないサーバに送信するように送信スレッドに指示する。

サーバでの処理実行間の遅延を避けるためはじめにサーバへ 2 演算ずつを送信しておく (図 3)。

この方法によりサーバではクライアントからの新たな処理実行命令の到着を待つこと無しに次の処理を実行できる。

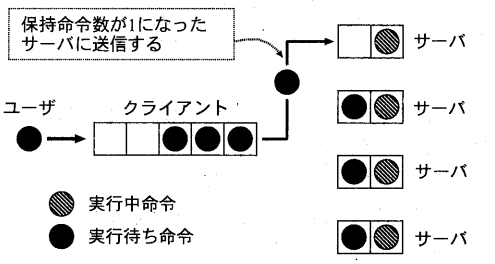

図 3: 計算時間に重点を置いた動的負荷分散方式の模式図

これによって処理実行が終了していないサーバはデータベースを更新しないため新たな命令が送られてくることがなく、またサーバでの処理実行間にタイムラグがを発生させることなくシステム全体の負荷を分散できる。

ネットワークを介したデータ移動時間を抑え全体時間に占める処理時間の割合を大きくするためにはサーバ間のデータ移動回数を減らす必要がある。そのため処理を実行するサーバに直接、処理実行に必要な引数データを送信しておくのが望ましいが、処理実行場所をスケジューラが決定しさらに決定するタイミングが不定であるため、処理実行サーバにデータをあらかじめ送信しておくことができない。

クライアントに一時データを保管しておく方法も考えられるがクライアントの主記憶領域を圧迫するため、ユーザからの要求があった時点で、データ保持数が最も少ないサーバにデータを送信する方法をとった。

サーバの格納命令数、データ個数は処理の実行状況にあわせてリアルタイムに変化していくので、それに合わせてクライアントのサーバ状態データベースも更新されていく必要がある。もしスケジューラが参照したサーバ状態データベースと実際のサーバの状態が食い違っていた場合、正確なスケジューリングが行われることは保証できない。そのためサーバからの状態送信用プロトコルには応答速度に重点を置く必要がある。

Clop ではデータや命令の通信プロトコルとして TCP を使用しているが、状態送信用プロトコルには通信速度を重視して UDP を採用した (表 1)。

TCP はデータの順序検査、エラー検査、再送処理などをプロトコルで規定しており信頼性が高い。しかし UDP では処理が単純で高速だがエラー検査、回復などはアプリケーション側で実装する必要がある [6]。本実装ではサーバからの状態送信データにはシーケン

プロトコル	時間 (sec)
TCP	0.0002594
UDP	0.0001587

表 1: 1バイトデータのピンポン転送に要する時間 (10回平均値、100BASE-TX)

スを付加し順序検査やパケット落ちをクライアントでチェックした。

4 Mandelbrot 集合作成時間の測定

本稿では動的負荷分散の効果を計るものとして、分散処理による Mandelbrot 集合作成時間を測定した。

$$z_0 = 0, z_{n+1} = z_n^2 - c(n = 0, 1, ...)$$

で定義される複素数の数列 z_n において $n \to \infty$ で $|z_n|$ が発散しないような複素数 c の集合 M

$$M = \{c \in C \mid \lim_{n\to\infty} |z_n| \neq \infty\}$$

を **Mandelbrot 集合**と呼ぶ [7,8]。

現実に計算機上で c が M に属しているかどうかの判定を行う場合、有限な計算量で行わなければならないためある繰り返し数を設定し判定を行う。図 4 に点 (x,y) での判定アルゴリズムを示す。

```
繰り返し回数をNとすると
z       ← 0
count   ← N
( ( |z| ≤ 4 ) かつ ( count ≥ 1 )なら繰り返す )
{
        z       ← z^2 - ( x + iy )
        count   ← count - 1
}
```

図 4: 点 (x,y) における Mandelbrot 集合判定アルゴリズム

x-y 平面上のある領域内の全ての点に対してこのアルゴリズムを適用し画像を作成する。

実験において描画領域を $(-0.6, -1.0) - (1.0, 0.0)$ にし、作成画像サイズを [480x480] に設定した。

実験は Mandelbrot 集合を作成する領域を列方向に分割し、サーバへサイクリック分配 (図 5) する方法と、今回実装した動的負荷分散を行いながらサーバへ分配を行う方法の 2 つを比較することで行った。実験はクライアント、サーバともに以下の環境で行った。

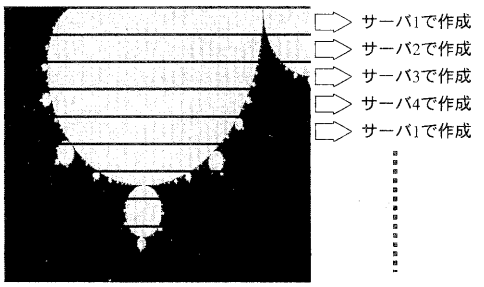

図 5: サイクリック分配による Mandelbrot 集合作成

使用機種	IBM PC/AT 互換機
CPU	PentiumII 333MHz
1 次キャッシュ	32Kbyte
2 次キャッシュ	512Kbyte
主記憶	256Mbyte
スワップ	120Mbyte
NIC	3Com 3c905B(100Base-TX)
スイッチングハブ	BayStack 450T Switch
OS	VineLinux 2.0 (Kernel 2.2.14)

4.1 サーバ 4 台による Mandelbrot 集合作成

サーバ 4 台を使用し繰り返し回数 300000 回の Mandelbrot 集合作成時間を計測した (図 6)。比較として、Clop を使用しない場合の作成時間は 1932.56(sec) である。

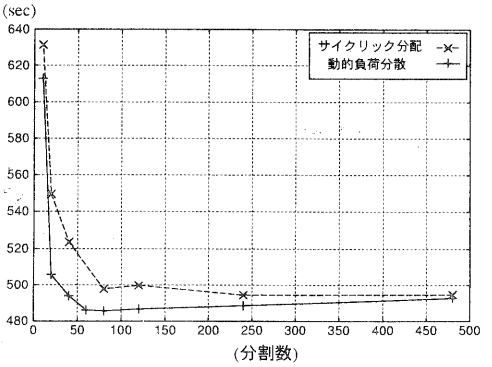

図 6: Mandelbrot 集合作成時間 (サーバ 4 台,300000 回)

分割数 40 から 120 付近で動的負荷分散を行った効

果が読みとれる。すべての点においてもサイクリック分配と比較して本稿で実装した動的負荷分散方式が短時間で処理が終了している。

分割数を大きくしていくと分割数 480 ではサイクリック分配との差は小さくなる。これは分割数を大きくしていくことで各サーバで実行される処理数も多くなるため、自然な負荷分散が行われているものと推察される。

逆に動的負荷分散では、分割数 480 の場合の処理時間が分割数 240 の場合に比較して大きくなっている。本稿での実装では処理実行に必要なデータをデータ数が少ないサーバに優先して送信している。しかし処理実行サーバを決定するのは処理の完了順番であり、処理実行サーバと必要データが存在するサーバが必ずしも一致しない。そのためサーバ間データ移動回数が多くなってしまっていることが原因であると思われる。

次に実際に負荷分散が行われているかどうかを示すために各サーバで行われた Mandelbrot 作成処理の合計時間を示す (図 7)。分割数は 40 である。

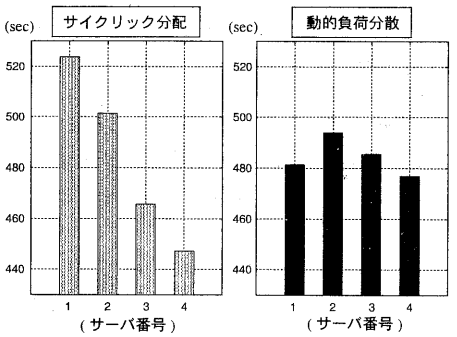

図 7: サーバ別 Mandelbrot 集合作成時間 (サーバ 4 台,300000 回,40 分割)

サイクリック分配を行った場合、サーバ 1 の負荷が最も高くなっており、これが全体処理時間の増加の原因であることがわかる。サイクリック分配を行った場合に比べて動的負荷分散を行った場合負荷の偏りが小さい。

4.2 サーバ 25 台による Mandelbrot 集合作成

サーバを 25 台使用した場合の繰り返し回数 300000 回での Mandelbrot 集合作成時間を示す (図 8)。

グラフから、分割数 60 以降は動的負荷分散を行った

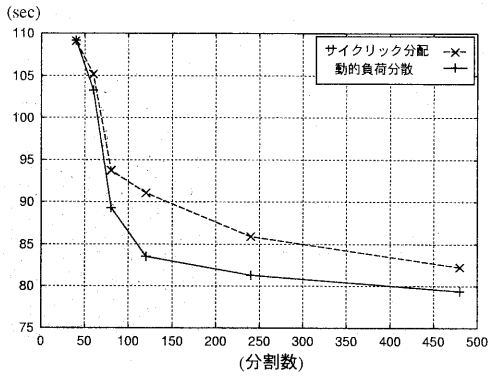

図 8: Mandelbrot 集合作成時間 (サーバ 25 台,300000 回)

場合の方が処理時間が短縮されている。その差は分割数 120 で最大となり、その後わずかずつ減少していっている。

分割数 120 での各サーバの計算時間の合計をグラフで表す (図 9)。

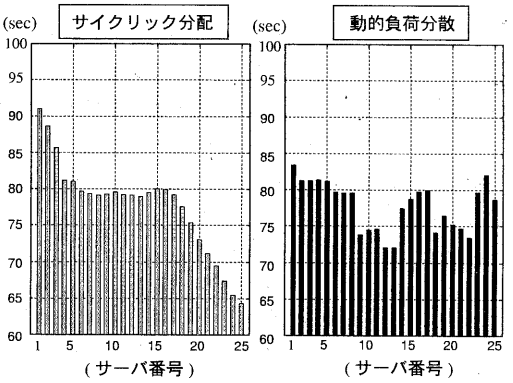

図 9: サーバ別 Mandelbrot 集合作成時間 (サーバ 25 台,300000 回,120 分割)

グラフから、サイクリック分配では負荷に大きな偏りがあるのに対して、動的負荷分散では負荷が小さい。

次に、分割数 480 での各サーバの計算時間の合計をグラフで表す (図 10)。

分割数 480 の動的負荷分散の場合、各サーバの処理実行時間がほぼ均等になっている。

サイクリック分配の場合では分割数 120 の場合に比べて負荷が分散されている。これから前述の、分割数を増加させたことによる自然な負荷分散が起こっていることが推察できる。

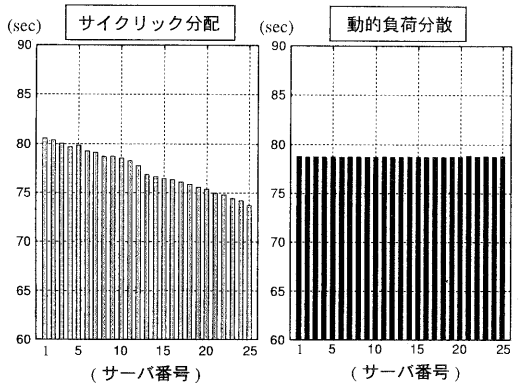

図 10: サーバ別 Mandelbrot 集合作成時間 (サーバ 25 台,300000 回,480 分割)

5 まとめ

クライアントにサーバの状態をリアルタイムに格納し、それをクライアントが参照しながら動的負荷分散を行うシステムを実装した。

さらに Mandelbrot 集合作成問題のような負荷に偏りがありユーザにとってそれぞれの負荷の予想が困難な問題において、システム側での動的負荷分散が有効であることを示した。

本稿では処理実行時間のみを基準にした動的負荷分散方式を実装した。しかし、計算機においての負荷は処理時間だけではない。分散処理による大規模行列の LU 分解などでは各計算機間でデータ量が大きい行列データを送受信させる必要があるため、主記憶量やネットワークトラフィックをもとにした負荷分散も検討する必要がある。

処理実行時間ではなくデータを中心に負荷分散を考慮した場合、Clop サーバへの処理の入力用引数データ転送は実際に処理を行うサーバに送信しておくのが望ましい。本稿で行った実装ではスケジューラが処理実行サーバを決定するためユーザはあらかじめ処理実行サーバにデータを送信しておくことができない。これを解決する方法としては、あらかじめ適当なサーバにデータを送信しておき、スケジューリングを行う際に処理実行終了メッセージ以外に、データ存在の有無も含めて処理実行サーバを決定する方法が考えられる。

今後は他のベンチマークも使用し、より詳細に性能を評価していきたい。

参考文献

[1] 松村 博光, 大鎌 広, 藤原 祥隆:“メッセージキャッシング型 PC クラスタにおけるスレッドの効果”, 情処研報告, 98-HPC-73-3,pp.13-18.(1998-10)

[2] 石澤 祐介, 大鎌 広, 藤原 祥隆:” 分散処理システム Clop におけるメモリ管理方式”, 情処研報告, 00-HPC-84-10,pp.55-60.(2000-10)

[3] 大鎌 広, 石澤 祐介, 藤原 祥隆:“メッセージプーリング:通信と計算を重ね合わせるクラスタコンピューティング方式の設計”, 情処研報告, 00-HPC-81-5,pp.25-30(2000-6)

[4] 村主 俊彦, 山下 博之, 木下 真吾:” 自律協調型分散システムによる負荷分散方法”, 電子情報通信学会論文誌, D-I Vol.J81-D-I No.10 pp.1115-1129.(1998-10)

[5] 天津 克秀, 平田 博章, 新實 治男, 柴山 潔:” 分散メモリ型並列計算機向き階層化スレッドスケジューリング方式”, 電子情報通信学会論文誌, D-I Vol.J80-D-I No.7 pp.615-623.(1997-7)

[6] W.Richard Stevens:”UNIX NETWORK PROGRAMMING”, Prentice Hall,Inc.(1990)

[7] 佐藤 光悦:” フラクタル/カオス 2”, ラッセル社 (1994-11)

[8] 上田 哲生, 谷口 雅彦, 諸澤 俊介:” 複素力学系序説 フラクタルと複素解析”, 培風館 (1996-11)