計算機アーキテクチャ 137－26
ハイパフォーマンス
コンピューティング 80－26
（2000．3．3）

PCクラスタ"Clop"におけるリアルタイムログシステムの設計

小林 賢一　大鎌 広　藤原 祥隆
北見工業大学

メッセージプーリング方式のPCクラスタを用いた分散処理システム"Clop"における、プログラム開発を支援するためのリアルタイムログシステムについて論じている。PCクラスタにおけるプログラム開発では、全計算サーバのイベント(計算開始、データ送信等)の実時間上の正確な順序を知り、かつそのイベントの種類と時刻の記録(イベントログ)を得ることが必要となる。しかしながら、分散処理している複数計算機は時刻が一致しているとは限らない。よって、イベントログを正確な順序で得ることは困難である。本稿では、ユーザのプログラム実行中に、計算機間の時刻のずれを計測することで、イベントログの発生時刻を補正できることが示されている。

Design of real-time logging system for PC cluster "Clop"

Ken-ichi KOBAYASHI　Hiroshi OHKAMA　Yoshitaka FUJIWARA
Kitami Institute of Technology

This paper presents a real-time logging system to develop programs on "Clop". "Clop" is a distributed computing system of Message-Pooling for a PC cluster. For developing the programs on the PC cluster, it is required of obtaining the exact sequence of events(starting of computation, sending of the data, etc.) and recording event log that includes type and time-stamp of the occurred events.

However, each clock of the computers on the PC cluster is not always equivalent to others. Therefore, it is difficult to obtain the event log with the exact sequence. It is shown to adjust the time-stamps in the event logs by the measuring offset of each of the computers during the execution of a user program.

1 はじめに

PCクラスタでの分散処理ライブラリとしてMPI[1,2]がよく知られているが、MPIを用いてプログラムを作成する際に通信部分の記述を行う必要がありプログラマの負担を大きくしている。そこで、筆者らは通信部分の記述で発生する負担を減らし、通信ブロックを軽減して処理効率を上げるための方式、メッセージプーリング方式(文献[3]では、メッセージキャッシング方式と記述)を実現したPCクラスタによる分散処理システムClop(Cluster of messeage pooling)の開発を行っている[3]。

しかし、プログラマが作成したプログラムを実行したり、デバッグするための環境に関しては、MPIではMPE[4]などの環境が用意されているが、Clopには開発環境と呼べるものがまだ用意されていない。ここでいう開発環境というものは次のような機能を提供するものである。

- イベント発生の監視
- 計算機の状態を監視
- ネットワーク状態の監視

Clop ではイベント (計算開始・終了、データ送信開始、データ受信終了等) の発生が分散して起こるので、それがプログラム開発を困難にする原因の一つとなっている。複数の計算機の時刻が誤差を有すること、またネットワークの遅れのため、イベントの実時間上の正確な順序を知り、そのイベントの種類と発生時刻を記録する (以後イベントログ) ことは難しい。

そこで、本研究で提案するリアルタイムログシステムでは、Clop の開発環境構築のために、計算機間の時刻のずれを計測することでイベントログの発生時刻が補正できることを示す。

2 Clop 概要

提案しているリアルタイムログシステムについて述べる前に、まずは分散処理システム Clop について説明する。

Clop の構成は、マルチサーバ・1 クライアントとなっている。ユーザは目的となる計算プログラムを作成する (これをユーザプログラムとする)。このユーザプログラムが実行される計算機を計算クライアントと呼び、それに対して、計算サーバプログラムを動作させる計算機を計算サーバと呼ぶ (図 1)。

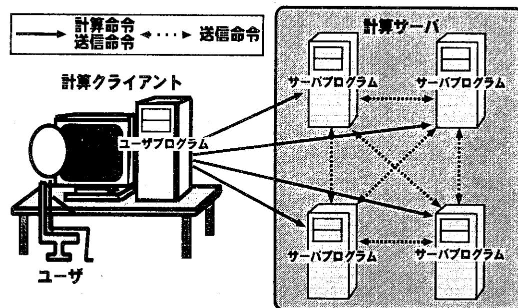

図 1: Clop システムの構成

実際の分散処理の基本動作は次の通りである。始めに全計算サーバを起動する。そして、計算クライアントでユーザプログラムを実行すると、ユーザプログラムは命令列に変換され、各計算サーバに送信される。そして、あらかじめ各計算サーバ内で起動されている計算サーバプログラムは、計算クライアントから送られてくる計算命令やデータを受信して、計算を行っている。最後に全計算サーバから計算クライアントに計算結果が集められる。このように、実際の計算は計算サーバが行っている。

Clop には次の二つの特徴がある。

- メッセージプーリング方式
- マルチスレッド化によるデータ送受信

2.1 メッセージプーリング方式

ネットワークを使用した分散処理で性能を向上させるために、計算と通信の重ね合わせを自動化したのがメッセージプーリング方式である。この方式では、データフロー型計算機システムのようにプログラムの記述順ではなくデータの依存関係によって実際の実行順序を決定している (図 2)。

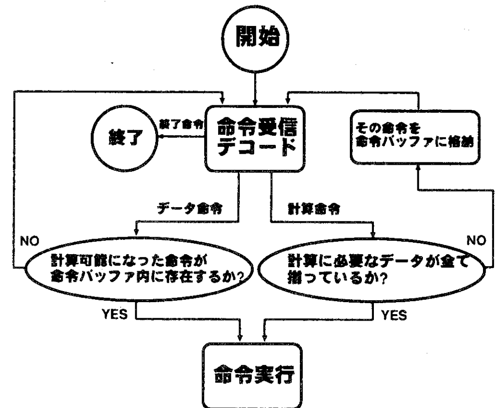

図 2: メッセージプーリング方式のアルゴリズム

計算サーバが受信する命令には大きく分けて 2 つある。一つは計算データが送信されてくるデータ命令で、もう一つは実行を指示する実行命令である。実行命令は、さらに計算を行う計算命令と他の計算機に計算データを送信する送信命令がある。計算命令の場合、受信時に必要な計算データが全て揃ってないことが考えられるので、実行できない命令を一時的に格納しておく命令バッファを用意する。データ命令を受信したら、命令バッファ内に実行可能になった命令がないか検索する。命令バッファを用いる事で、計算データの不足による実行命令が実行不可能な場合でも、すぐに次の命令受信に移って実行可能な命令から順次処理していくことができる。

2.2 マルチスレッド化によるデータ送受信

PC クラスタにおいて、計算機間の通信は内部のデータ転送に比べ明らかに遅い。しかし、サーバプログラム内の通信部分と計算部分を並列に処理することができれば計算機間の通信の遅さを隠蔽できる。そこで、

スレッド [5] を用いた方法で、サーバプログラムは並列処理を行う構成となっている。

3　Clop における開発環境の問題点

3.1　サーバプログラムの起動

Clop では、計算クライアントでユーザプログラムが実行される前に、全計算サーバでサーバプログラムが起動していなくてはならない。これまでは次の方法を用いてきた。

- 全サーバ数分の端末を用意して、それぞれサーバプログラムを起動
- シェルスクリプトで一括起動

全サーバ数分の端末を用意することは、サーバ数が増えてくるとそれだけでユーザの大きな負担となり得る。しかし、一つの端末上で全サーバを一括起動するようなシェルスクリプトを実行した場合、起動した端末画面上には複数の計算サーバからのイベントログが混在してしまう。これでは、必要なイベントログを判別できなくなる。

3.2　非決定的な実行順序

サーバプログラムでは、メッセージプーリング方式を用いることで通信によるブロックを軽減している。命令を受信した時点でその命令が実行可能かどうかを判断し、不可能であれば命令バッファに格納している。よって、計算クライアントが送信した順序で命令を実行している訳ではない。そのため、計算中に計算クライアントの計算進行状況を監視していても、実際の計算状況の把握が困難である。

また、計算サーバ内ではスレッドが複数存在しており、それらは並行して動作しているため、サーバ内の計算状況を知る際のプログラマへの負担が大きくなる。

3.3　時刻のずれ

Clop を構成している全ての計算機の時刻は一致してるとは限らない。そのため、命令実行やデータ送受信のイベント発生時刻が正確でなくなり、処理全体の流れが正しく判断できなくなる。

4　ログシステム

Clop の開発環境構築を目指した場合、まずは確認されている問題点を解消していかなければならないと考えた。そのために必要な機能として、

- 全計算サーバの一括起動
- どの計算サーバからでもイベントログを受信可能
- 計算中に全計算サーバの実行状況を確認できる
- 時間のずれの補正
- イベントログの表示

が挙げられる。これらの機能を実装することにより、プログラム開発の際にプログラマにかかる負担を軽減し、作業効率を上げることができる。そこで、これらの機能を有したシステムを設計した。これを、**リアルタイムログシステム**と呼ぶことにする。

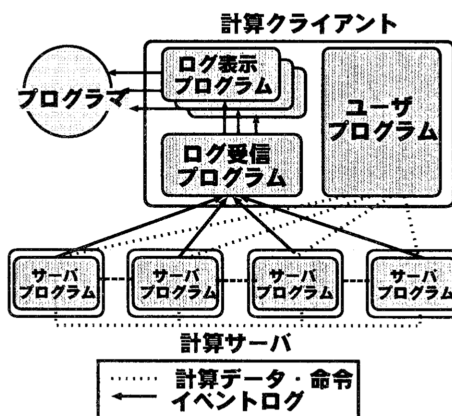

図 3: ログシステムの構成

4.1　構成

リアルタイムログシステムは、計算サーバを一括起動して全計算サーバからイベントログを受信するためのプログラム (**ログ受信プログラム**) と、イベントログを画面に表示するためのプログラム (**ログ表示プログラム**) の二つで構成されている (図 3)。

ログ受信プログラムの役割には、次のようなものがある。

- 全計算サーバの一括起動
- 全計算サーバからのイベントログをまとめて管理
- 各計算サーバとの時刻を補正する
- ログ表示プログラムへイベントログデータを送信

全計算サーバからのイベントログをまとめて管理することで、全てのイベントログを監視することが可能である。そのためプログラマはクライアント側でサーバのログを混在することなく取得できる。

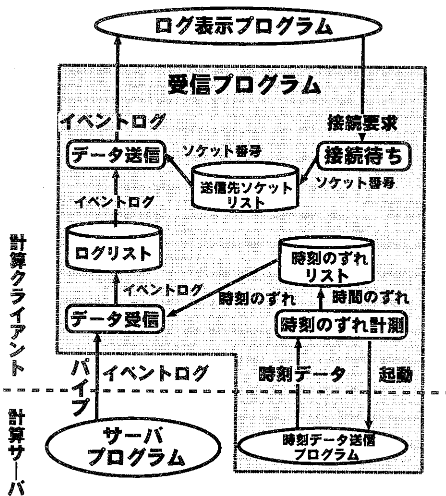

図 4: ログ受信プログラムの動作

また、ログを表示する機能をログ表示プログラムに任せることで、プログラマが調査すべきイベントに応じたログを取得できるようになる。これは、ログ表示プログラムを複数起動して、ログ表示プログラムで表示させるイベントを変えることができるからである。

4.2 ログ受信プログラム

このプログラムの動作を図 4 に示す。まず全計算サーバを起動し、各計算サーバ別にパイプ [6] を作成する。これにより、ログがどのサーバから送信されたのか、受信する際にパイプで判別することができる。また、サーバ別にパイプが存在するので、イベントログの混在を防ぐことができる。そして、各計算サーバとの時刻のずれを計測する。これは、サーバ内のイベントログ発生時刻を正確にするためである。イベント発生時刻をクライアントの時刻に合わせて補正することで、システム全体の処理の流れを把握できるようになる。時刻補正については後節で詳説する。

イベントログデータにはイベントログの発生した時刻 (タイムスタンプ) や種類と、プログラマが記述したコメントが格納されている。サーバプログラムはこのイベントログデータを送信する。

次に、サーバプログラムからイベントログを受信する。これは計算中に、いつ、どのサーバからでも受信できるようにしなければならない。それは送られて来るイベントログが非決定的であるからである。

イベントログを受信している間も並行して、接続したログ表示プログラムへイベントログデータを送信している。またこのプログラムでは、新たなログ表示プログラム接続要求とサーバプログラムからのイベントログ受信の、非決定的な順序の要求を受け付けなければならない。そのため、通信待ちによる処理効率の低下が起こる可能性がある。これを防ぐためにスレッドを用い、それら二つの要求を並行に実行することにした。

4.3 ログ表示プログラム

このプログラムはログ受信プログラムと分けられている。ログの表示部分をサーバ起動端末と分離することで、複数の端末にイベントログを表示できるようになる。これは Clop において、次に挙げたイベントの場合に有用である。

- 命令送信開始
- 計算データ受信時の計算実行
- 命令受信時の計算実行
- データ受信待ちの命令を命令バッファに格納

これらは計算サーバ内の処理で非決定的であり、処理の流れを把握するために重要なものだからである。複数の端末に異なるログを表示可能なので、上で挙げたイベントに関係するログをまとめて表示する、といった使い方ができる。

5 時刻補正

ログ受信プログラム内で、**時刻補正**を行っている。これは Clop システム全体の状況を知る上で大変重要である。なぜなら、計算機間では時刻のずれが生じるからである。

ログを取得する場合、一台の計算サーバ内のイベントログの順序は保証されるが、複数の計算サーバからログを受け取った場合に、それらのログの順序は保証されない。なぜならネットワークで接続されたサーバは並列に動作しているからである。たとえ、あるサーバがイベントログを他のサーバより早く送信したとしても、通信間に別のサーバが後から送信したイベントログがクライアントに受信される可能性もある。これは、ネットワークに接続してデータの送受信を行うため、遅延が発生するからである。また、たとえイベントログの順序入れ換えが無い場合でも、計算機間の時刻が一致していなければ、実時間上の正確な計算実行状況をログとして記録・表示することは不可能である。そこで、Clop の全体状況をリアルタイムで把握するために時刻補正を行う機構が必要である。この時刻補正

は、計算機間で時刻データを送受信し、お互いの時刻データを比較する。そして、その差を時刻のずれとして記録し、リストに格納する際にイベントログ内のタイムスタンプをその時刻のずれを用いて補正することで行われる。

MPI の開発ライブラリである MPE では計算終了後に時間合わせを行うことで、ログの順序を保っている。しかし、リアルタイムログシステムではユーザプログラム実行中に時刻補正を行うことで、イベント発生直後に処理状況を確認できることを目指している。

5.1 遅延時間

時刻補正を行う際に、時刻データの送受信を行う。その時、時刻データが送信されてから、ネットワークを経由して受信されるまでには時間が経過する。この経過した時間を**遅延時間**という。遅延時間が分らなければ、計算機間で時刻データの正確な比較を行うことが困難になる。そこで時刻補正を行うために、遅延時間を計測する。その計測方法として、単純に 2 台の計算機間でピンポン転送を行うことにした。この動作は図 5 に示す。また、ピンポン転送に UDP[6] を用いている。

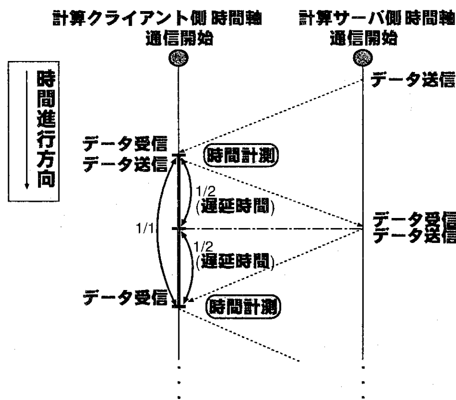

図 5: ピンポン転送の動作

互いにデータの送信と受信を交互に行い、その間クライアント側はデータ受信時間を記録しておく。そして連続した受信時間の差を 2 分の 1 にした値を遅延時間とした。例として LAN(100BASE-TX) で接続された PC(PentiumII 333MHz, 128MByte) 間でピンポン転送の転送数を変えて計測した結果が図 6 である。

このグラフは計測した遅延時間を、昇順にソートし

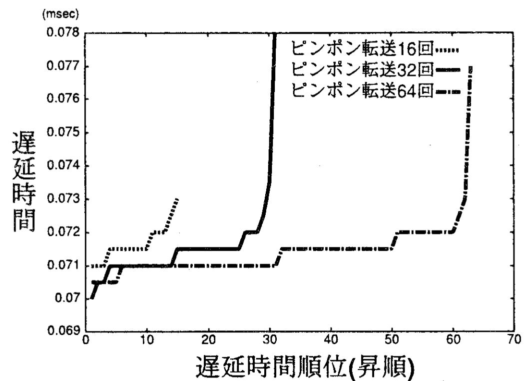

図 6: 遅延時間の計測

てグラフ化したものである。グラフを見ると、ピンポン転送の数に関係なく順位の上位値と下位値の数個が突出しており、異常データが存在している。時刻補正に用いる遅延時間は誤差が小さい方が良いが、このままだと誤差が大きくなることが予想される。そこで、計測値をそのまま平均した値と、異常データを捨てて計測値の誤差を 0.001msec 以内にした場合の平均値を計算した (表 1)。

表 1: 遅延時間の平均値 (msec)

転送数	異常データ含む	異常データ含めない
16 回	0.07166	0.07150
32 回	0.07156	0.07108
64 回	0.07142	0.07132

異常データとみなした個数は、32 回のピンポン転送で 6 個、64 回の場合で 7 個であり、これ以上多くのピンポン転送を行う必要は無いと思われる。

5.2 時刻差

実際に時刻補正をするためには、サーバ・クライアント間の時刻のずれ (以後、**時刻差**とする) を求める必要がある。そのために、それら 2 台の時刻を比較する。いま、次のように仮定する。

- クライアント A とサーバ B がある
- 比較開始 t_0 時における、A のローカル時刻を $C_a(t_0)$、B のローカル時刻を $C_b(t_0)$ とする
- $C_a(t_0)$ は直接計測できないので、t_0 から遅延時間 d 後に計測し、d を引く

$$C_a(t_0) \cong C_a(t_0 + d) - d$$

- 比較開始から n 分後の時刻を $C_a(t_n)$、$C_b(t_n)$ とする

このとき、A からみた B との n 分後の時刻差は

$$I_{ab}(t_n) = C_a(t_n) - C_b(t_n)$$

で求められる。遅延時間測定に用いられたものと同じ構成の PC サーバ 4 台の時間差を計測した (図 7)。

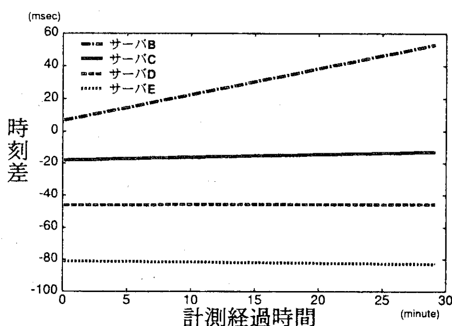

図 7: クライアント A とサーバ 4 台の時刻差

グラフから、時刻差の変化値が経過時間に比例することが分る。それにより、時刻差の変化予測が可能である。つまり、分散処理前にログ受信プログラム内で一度時間合わせを行う。そして、時刻差の許容範囲を決めておけば、ユーザプログラム実行中にその範囲を越える時があった場合に再度時刻合わせを行えば、時刻差はその許容範囲内に抑えることができる。

5.3 予測誤差

クライアント A からみたサーバ B の時刻差 $I_{ab}(t_n)$ を計測後、1 分間での時刻差変化の比例係数を、

$$\alpha_{ab} = \frac{I_{ab}(t_n) - I_{ab}(t_0)}{t_n - t_0}$$

とする。n 分後のクライアント A とサーバ B の予測時刻差の値は

$$\hat{I}_{ab}(t_n) = I_{ab}(t_0) + \alpha_{ab} t_n$$

である。この予測時刻差と、予測後に計測した時刻差との差を**予測誤差**とした。比例係数 α は 30 分間の計測データを用いて計算し、図 7 で用いた 4 台のサーバを再度使用して予測誤差を計測した (図 8)。

サーバ D・E は予測誤差を 2msec 以内に抑えることができたが、サーバ B・C に関しては 5msec 以上の予測誤差が発生した。この結果では、12 時間で最大 23msec の誤差がみられた。時刻差の変化値が大きい計算機の場合にはより長時間の測定データが必要なことが分った。

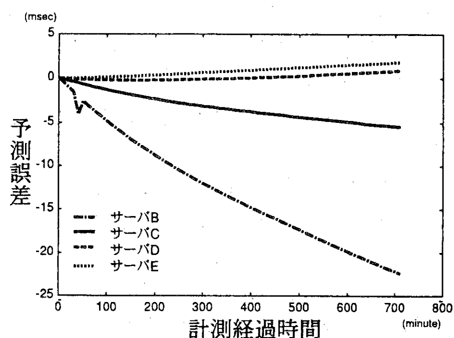

図 8: クライアント A とサーバ 4 台の予測誤差

6 まとめ

今回の結果から、時刻補正を行ったことで計算機間の時刻のずれを抑えることができた。しかし、時刻補正の場合には、事前に予測するための時刻差計測データをある程度の時間分用意しなければ、予測時刻差の信頼性が低くなってしまうことが分った。

また、遅延時間の測定では突出した値を計測することがあり、このデータを除いて時刻補正を行った。

今後は時刻補正を用いて、Clop でのプログラム開発に有用な環境を構築することを目指している。

参考文献

[1] D.Snir, S.Otto, S.Huss-Lederman, D.Walker, J.Dongarra:"MPI-The Complete Reference Volume 1,The MPI Core secondedition", The MIT Press.(1998)

[2] W.Gropp, S.Huss-Lederman, A.Lumsdaine, E.Lusk, B.Nitzberg, W.Saphir, M.Snir: "MPI-The Complete Reference Volume 2,The MPI Extensions", The MIT Press.(1998)

[3] 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)

[4] W.Gropp, E.Lusk, A.Skjellum:"USING MPI", The MIT Press.(1994)

[5] B.Nichols, D.Buttlar, J.P.Farrell :"Pthreads Programming", O'REILLY.(1998)

[6] W.Richard Stevens:"UNIX NETWORK PROGRAMMING", Prentice Hall,Inc.(1990) 日本語訳, 篠田 陽一, 株式会社トッパン