ハイパフォーマンス
コンピューティング 84−11
（2000．12．8）

分散処理システムClopにおけるロギングシステム用インターフェース

小林 賢一　大鎌 広　藤原 祥隆
北見工業大学

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ロギングシステムを提案し、そのインターフェースを論じている。分散処理システムClopはシングルクライアント・マルチサーバ型のPCクラスタ用のシステムとして開発されている。Clop上のプログラムのデバッグ、チューニングには、全サーバ内で発生したイベントの種類と時刻の記録(イベントログ)を得ることが要求される。しかし、イベントログを管理するには、各プログラムでログの形式を統一する必要がある。本稿では、イベントログデータ用クラスとイベントの発生場所や種類を指定可能なイベントログインターフェースを設計する。これにより、必要なイベントログを抽出して表示可能になる。

Interface of a logging system on distributed computing system Clop

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

A logging system to support program development on Clop is suggested. An interface of the logging system is discussed. Clop is developed for a distributed computing system on a PC cluster of single-client multi-server. For debugging and tuning the programs on the Clop, it is required of obtaining event logs that include type and time-stamp of the occurred events. It is necessary to unify a form of event logs in the programs in order to manage the event logs. In this paper, C++ class of the event logs is designed. The interface for the event logs can specify type and place of the occurred events. The logging system enables to extract and display the necessary event logs.

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を用いて高速処理を実現する際に通信に関する記述を詳細に行う必要があり、プログラマの負担を大きくする原因となっている。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[3,4,5]。

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[6]などの環境が用意されているが、Clopにおける開発環境は開発中である[7]。

そこで、本稿で提案するロギングシステムでは、Clopの開発環境構築のためにイベントログ出力用インターフェースを設計・実装し、デバックやチューニングを目的としたイベントログの出力や表示に関するプログラマの負荷を軽減することを示す。

2　開発環境の問題点

Clopを利用する場合、使用する全計算機上でユーザが作成したユーザプログラムを起動して実行する。

起動させるユーザプログラムは全計算機で共通のものだが、各計算機で設定ファイルを読み込み、ユーザプログラム内で、クライアントか、もしくはサーバかを判断し、それぞれに対応したルーチンを呼び出している。そこで、クライアントで起動しているユーザプログラムを **Clop クライアント**、各サーバで起動しているユーザプログラムを **Clop サーバ**とする (図 1)。また、以後本稿では Clop を利用するユーザと Clop システムの開発者を総じてプログラマと呼ぶ。

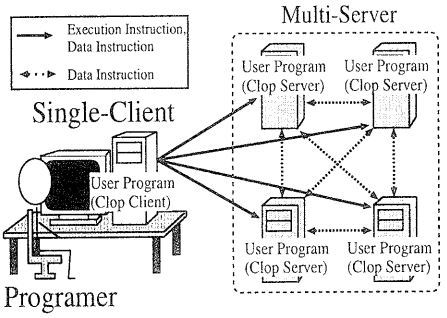

図 1: Clop システムの構成

Clop サーバ内で発生しているイベントをプログラマが監視するためには、Clop サーバでイベントログを出力して確認しなければならない。このとき、プログラマは全 Clop サーバのイベントログを判別して、プログラム開発に必要な情報を取得することになる。そうなると、必要としている情報のみを表示することが要求されるが、現状では次の問題がある。

- 使用サーバ数が増えると、イベントログの量が増加
- イベントの正確な発生順序を得るための負担が大きい

高速処理を目指すために使用するサーバ数を増やすことで、イベント発生量が増えてしまうので、本当に必要な情報を探すことが難しくなる。正確な発生順序を得るための時刻設定については文献 [7] で報告し、現在もそれをもとに改良中である。本稿ではイベントログの集積、必要な情報の表示、ログ用インターフェースに焦点をあてる。

3 ロギングシステム概要

Clop 上でプログラム開発を行う際に、プログラマにかかる負担を軽減して作業効率の向上を支援する開発環境構築を目指している。

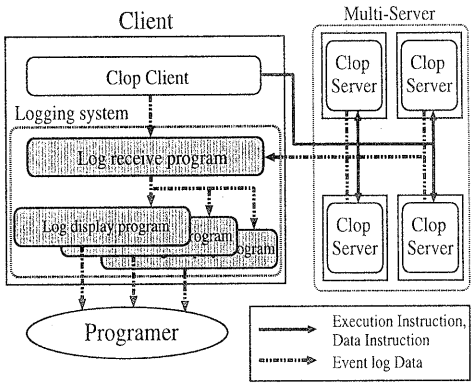

図 2: ロギングシステムの構成

Clop では、Clop クライアントを実行すると、ユーザが作成したアルゴリズムに従い Clop サーバのリモート関数を呼び出す。あらかじめ起動している Clop サーバは Clop 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。このように実際の計算は Clop サーバが行っている。

そこで、全計算機でのユーザプログラム起動、またイベントログ取得等の作業を全てクライアントマシン上で実行可能にすることで作業工程を短縮し、イベントログ表示の際には、指定したもののみを画面上に表示するようにする。そうすることで、負担を軽減し、作業効率を高めようと考え、これらを実現するシステム、Clop における**ロギングシステム**を設計した。

3.1 構成

ロギングシステムは、全 Clop サーバから出力されたイベントログをクライアント側で受信するためのプログラム (**ログ受信プログラム**)、そして、イベントログを画面上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 2)。これらのプログラムは Clop クライアントが実行される計算機で実行されることになる。Clop クライアントと全 Clop 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全ての Clop 利用マシンのイベントログを取得できる。そして、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項目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可能で、表示させるイベントログをそれぞれ変える

ことができるからである。

3.2 ログ受信プログラム

ログ受信プログラムの役割は、次に示す通りである。

- 全 Clop サーバの一括起動及び終了
- Clop クライアントの実行
- 全 Clop サーバからのイベントログをまとめて管理
- クライアントと各サーバ間の時刻のずれを計測する
- ログ表示プログラムへイベントログデータを送信

この中で Clop サーバの起動及び終了と、Clop クライアントの実行は、プログラマの作業簡略化を目指したものである。従来の方法ではこれらの作業負荷が問題となっていたが、ロギングシステムを用いると、Clop で使用する全計算機でユーザプログラムが自動的に実行される (図 3)。

このプログラムの動作は下記の通りである。まず起動時に Clop 個人設定ファイルを読み込む。このファイルからログ受信プログラムは Clop サーバを起動するホスト名とその数を取得する。

時刻のずれの計測はずれの値を全サーバ数分を保持しておき、イベントログデータ受信時にイベントログ発生時刻をクライアントの時刻に合わせて補正するために行う。これにより、ログが表示される際に Clop 全体の処理の正確な流れが把握できるようになる。

イベントログデータを受信している間も、ログ表示プログラムへイベントログデータを送信しなければならない。そこで、ログ受信とログ送信を並行に行うためにスレッド [8] を用いた。

3.3 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。なぜならログの表示部分を分離することで、複数の端末にイベントログを表示できるようになる。また、それぞれの端末上で個別に表示するイベントを特定することができる。

これにより、プログラマが任意にログ情報を分別することが可能になるので、プログラム動作把握の負荷を軽減することが期待できる。

ログ表示プログラムは次の役割がある。

- ログ受信プログラムからイベントログデータを受信
- プログラマが指定した必要なイベントログを分別する

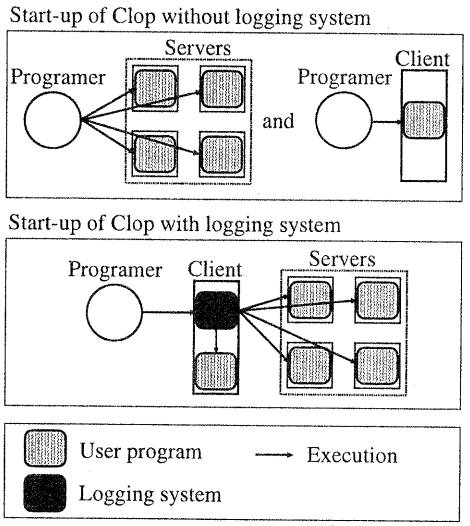

図 3: Clop 起動の自動化

- 指定されたログのみを発生時刻順に表示

このプログラムは、受信したイベントログデータを格納し、プログラマが表示指定したものかどうかを判別する。もし、表示指定されたものであれば、バッファに蓄えておく。一定時間バッファに溜ったイベントログデータを発生時刻でソートし、画面に出力していく。これは、イベントログデータが Clop クライアント又は Clop サーバで出力されてから、ログ表示プログラムに受信されるまでの順番が保証されていないので、イベント発生順序を整理してプログラム動作の把握を容易にするためである。

3.4 ロギングシステムの問題

ロギングシステムを実装する上で問題がある。

Clop サーバ、Clop クライアント共にイベントログデータを出力するが、このデータは共通のデータ形式でなければ同等に管理不可能である。また、共通のデータ形式を実現した場合、プログラマはイベントログデータを出力するためのインターフェースが提供されていなければ出力は困難である。

これまで、イベントログの出力は printf 関数で行っており、このログの中にはイベントの発生場所や種類が含まれている事が分かっている。そこで、標準的にその二つの情報を取得できる機能が必要である。

4 イベントログデータ出力

前節で述べた問題は、プログラマがイベントログの操作を全て自らが行わなければならない事が原因である。それを解決するために、イベントログ操作を統一化と、自動化することが可能な仕組みを用意する。そこで、イベントログ用のデータクラスを設計し、そのインターフェースを提供する。提供するイベントログデータ用の内部インターフェースは、問題を解決するために次に挙げる機能が必要である。

- Clop クライアント内の任意の場所でイベントログを出力可能
- Clop サーバ内でも同様にイベントログを出力可能
- イベントログデータに発生時刻を添付する機能
- スレッド間でのログデータロック機能
- イベントログデータの種類を特定可能

プログラマは、このインターフェースを用いて Clop の全プログラム内でイベントログを出力する。それにより、イベントログの操作を統一化し、不要な手順を省略することが可能となる。また、Clop サーバ内には複数のスレッドが常時動作しているので、同時にイベントログを出力してデータが混在する危険性があるのでロック機能は必要である。

4.1 CE_eventlog クラス

Clop システムは C++によって実装されている。そこで、インターフェースもこれに準じることにする。イベントログデータの操作方法を共通化するために、イベントログデータ用のクラスを設計した。このクラス (以後、CE_eventlog クラス) の実装を以下に示す。

```
class CE_eventlog {
  private:
    int       place;    // ログ特定用変数
    int       type;     // ログ特定用変数
    int       msize;    // message の文字数
    char     *message;  // ログメッセージ格納
    struct timeval timestamp;
                       // ログ発生時刻
  public:
    // メンバ変数を操作するための関数群
};
```

メンバ変数 place と type は、出力の際にプログラマが指定した place と type によってイベントログの発生場所や種類を特定するために実装されている。指定方法は直接値を指定するのではなく、後節で詳説するインターフェースの引数として、それぞれの値に対応した文字列 (キーワード) で行う。place にはクラス名や、ファイル名、スレッド名等のイベント発生場所を指定し、type には発生した種類 (データ送信、計算実行等) を指定する。次に、メンバ変数 message はプログラマが付属できるイベント固有のメッセージである。その文字数は message 格納時に自動でメンバ変数 msize に格納される。最後のメンバ変数 timestamp はイベントが発生した時刻を格納する。

CE_eventlog クラスはイベントログ情報に必要なデータを格納し、それらのデータ操作を提供することが目的である。結果的に開発環境を構築する際、イベントログデータの操作を容易にするが可能となる。

4.2 プログラミングインターフェース

CE_eventlog クラスを用いて、プログラマがユーザプログラムや Clop システム内からイベントログを出力するためのインターフェースを設計した。

```
int CE_logoutput(
      const char *sp, // ログ特定用引数
      const char *st, // ログ特定用引数
      char  *logm,    // ログメッセージ
      ... );
int CE_logerrput(
      const char *sp, // ログ特定用引数
      const char *st, // ログ特定用引数
      char  *logm,    // ログメッセージ
      ... );
```

両関数とも引数は共通で、イベントを特定するための文字列 sp、st を、イベント固有の任意のメッセージ logm を引数として受け取る。そして、ログ発生時刻を添付した状態で、CE_logoutput() は標準出力へ、CE_logerrput() は標準エラー出力へイベントログデータを出力する関数である。この二つの関数は可変長引数となっており、logm 以降の引数は printf 関数と同じ書式で書くことができる。これは、既存の printf 関数と使用感を近づけることで、プログラマの利便性を高めることが可能と考えたためである。戻り値は両関数共に、実際に出力したバイト数である。これらの関数内では、CE_eventlog クラスのデータを宣言し、イベントログ特定引数 sp と st をそれぞれ対応したユニーク

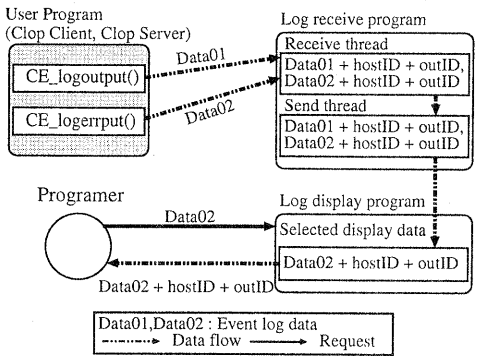

図 4: イベントログデータの流れ

な定数へ変換し、メンバ place と type に格納する。また、発生時刻を計測して timestamp へ格納、メッセージ logm は整形後に message へ格納し、出力している。

ログ受信プログラムはこの出力されたイベントログデータを受信し、出力元ホスト番号 (hostID)、出力形式番号 (outID) と共にログ表示プログラムへ送信する (図 4)。出力形式とは標準出力か標準エラー出力かを区別するものである。また、ホスト番号はロギングシステム内で定義した各ホストに付けたユニークな値である。

これらの値を CE_eventlog クラスに組み込まず、ログ受信時に追加する理由は次の 2 点である。

- 計算機間の通信データ量を抑えるため
- ログ受信プログラム内で判別可能であるため

計算機間の通信データ量は、Clop 全体のパフォーマンスに影響する可能性があるので、イベントログデータにおける通信負荷を抑えた設計にした。

プログラマはこれら二つの関数を用いることで、ユーザプログラム、Clop システム内において、同様の方法でイベントログ出力を行うことができ、出力されたイベントログデータはロギングシステム内で管理され、必要なイベントログのみを特定し、表示することが可能となる。そのとき、イベントログをイベント特定用文字列 sp や st、又は発生時刻によって指定して、絞り込むことが可能である。

5 ロギングシステム使用例

例として、ユーザプログラム linpack00 を計算機 clop01、clop02、clop03、clop04、clop05 の 5 台を使用して実行する。また、clop01 を Clop クライアントとし、他の 4 台を Clop サーバとするように設定ファイルで記述されているとする。

始めに、clop01 上でログ受信プログラム CE_logrecv を実行して、全利用 Clop サーバの起動と Clop クライアント実行を行う。

```
> CE_logrecv linpack00
```

いま、clop01 内で標準出力へイベントログを出力したとする。その記述は

```
CE_logoutput("CLI","DSEND","ID %d",i);
```

である。また、同様に clop04 内で標準エラー出力へイベントログを出力するために次のように記述したとする。

```
CE_logerrput("SERV","DRECV","ID %d",i);
```

この場合、プログラマは CLI と SERV、DSEND と DRECV にそれぞれ対応したログ表示の際、実際に画面表示される文字列を指定しておくことが可能である。その指定は次のような対応表が記されたファイル内でプログラマが指定しておかなければならない。

```
// 記述形式
// 特定用文字列 : 実際の表示文字列
        CLI   : Client
        SERV  : Server
        DSEND : SendData
        DRECV :
```

このファイルは Clop クライアントと Clop サーバ、そしてロギングシステムの起動時に読み込まれる。

i の値が 2 であった場合、上の二つの関数が実行された結果は、ログ表示プログラム CE_logdisplay を実行すると次のように画面に表示される。

```
> CE_logdisplay
clop01:out:Client:SendData:ID 2
clop04:err:Server:DRECV:ID 2
```

ここで、DRECV は表示時の文字列がファイルに記述されていないので、そのまま表示されている。もし、対応記述ファイル内で DRECV の記述が無い場合は、次のように表示される。

```
clop01:out:Client:SendData:ID 2
clop04:err:Server:NONE:ID 2
```

このように画面に表示したいイベント特定文字列をプログラマが任意に指定することができ、それを用いて表示したいログデータを指定することが可能である。また、それぞれの項目 (ホスト名、出力形式、特定文字列、メッセージ、発生時刻) 毎に表示・非表示を CE_logdisplay の引数で指定可能である。

6 システムの有効事例

ロギングシステムは、使用しないときに比べ、次のような状況において有効である。

- 実行が長時間に及ぶ場合
- 実行中に必要な情報が変化した場合

ロギングシステムを使用しないでイベント情報を取得する場合、データ保持の状況やメモリ使用状況、又は待機時間等の種類に特化した情報を調べるためには、計算終了後に出力したログファイルを使用しなければならなかった。しかし、ロギングシステムを使用すれば、プログラム実行中にイベントの発生場所や種類を特定して表示することが可能である。つまり、実行中にあるサーバがデータ待ち状態で止まってしまった場合、どのサーバがそのデータを保持しているか調べる必要がある。そこで、データの送受信とデータ保持に関するイベントログのみを表示するために新たにログ表示プログラムを実行する。これだけで、いま求めているデータに関連する情報をそのサーバがデータ待ち状態のなか、取得できるということである (図 5)。

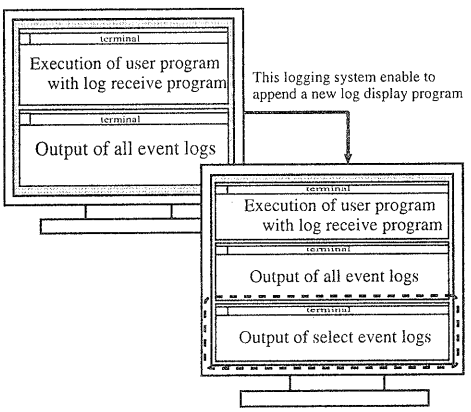

図 5: ユーザプログラム実行中のログ表示プログラム追加例

7 まとめ

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に、ユーザプログラム内でイベントログのデータ形式を統一化し、イベントログ出力用のインターフェースを設計・実装した。イベントの発生場所や種類をプログラマが指定可能な設計にすることで、プログラマが必要とするイベントの情報を抽出し、表示する方法を提案した。また、そのインターフェースとロギングシステムを使用することで可能となったイベントログ表示方法を示した。従来では不可能であった実行中のイベント情報の切替えを可能としたことで、Clop プログラム開発におけるデバッグやチューニングの効率が向上する可能性を高めた。

しかし、イベントを詳細に指定可能にするためには、プログラマが表示される文字列を決定し直接ファイルに記述しなければならない。また、実際に指定する際もログ表示プログラム実行時にその引数として与えるしかない。これらの点については、Clop が対象としているユーザに対してより良い環境を提供する必要がある。

今後は、ログの指定方法の改善について研究を行い、より有用な Clop における開発環境の構築を目指したい。

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)

[4] 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, Vol.2000-HPC-81-5,pp.25-30(2000)

[5] 石澤祐介, 大鎌 広, 藤原 祥隆:PC クラスタ"Clop" の LINPACK Benchmark による性能評価, 並列処理シンポジウム JSPP2000, Vol.2000, No.6, p.163(2000)

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

[7] 小林賢一, 大鎌 広, 藤原 祥隆: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, 情処研報告, Vol.2000-HPC-80-26,pp.149-154(2000)

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