マルチメディア通信と分散処理　102－26
（２００１．３．２２）

種々の並列・分散アプリケーションに対して容易に統合可能な動的ロードバランサ pDLB の提案と実装

潤田浩也　弓場敏嗣　佐藤直人†

電気通信大学大学院　情報システム学研究科
†現在　日本アイ・ビー・エム株式会社

計算機クラスタを用いた分散環境が並列・分散処理の有効なプラットホームとして注目されている。クラスタを構成する個々のプロセッサの負荷変動が大きい分散環境では、並列・分散アプリケーションプログラムを実行する場合、動的な負荷分散を行なうことが必要となる。しかし、アプリケーションプログラムに負荷分散コードを組み入れることは一般に複雑で手間がかかる。また、逐次プログラムを並列・分散プログラムに改変するとき、変更部分を最小限にとどめ、かつ可搬性を保つことが望ましい。さらに異なるアプリケーションに対して同じ負荷分散方法を適用し、負荷分散を行なうスケジューリングプログラム自体の再利用性を高めることが望ましい。本論文では、動的ロードバランサ用ライブラリを用いて並列・分散アプリケーションを記述することによって動的負荷分散を達成する方式を提案する。アプリケーションから独立し、かつ容易に統合できる動的ロードバランサを実装し有効性を評価する。

pDLB : A Portable Dynamic Load Balancer Easily Integrated to Various Distributed Applications

Hiroya Uruta , Toshitsugu Yuba , Naohito Sato †
The Graduate School of Information Systems, University Of Electro-Communications, † IBM Japan.Ltd

Cluster computing environment is received much attention as an effective platform of parallel processing. It is necessary to perform a dynamic load balancing when execute parallel program on this environment. However, generally integrating a load balancing code into application program is complicated. And it is desirable that modification from a serial program to a parallel program is minimized as much as possible and at the same time the portability still remains. Furthermore, when applying the load balancing method for different parallel application it is also desirable to increase reusability of the scheduling program. In this paper, we propose a novel way which uses a dynamic load balancer library pDLB that is independent from the application and is easily integrated to application program. We have implemented this pDLB and evaluated its effectiveness.

1　はじめに

近年、計算機技術の向上により複数の要素プロセッサによって構成された並列計算機が商用化され、一般に手に入れやすいものになった。また、myrinet[1]等の高速なネットワークの出現に伴って、複数の高性能ワークステーションやパーソナルコンピュータを高速ネットワークで結合した

計算機クラスタが、安価で使い易い並列・分散処理の有効なプラットホームになっている。ネットワークの高速化に伴って並列・分散処理はますます身近なものになり、広く普及していくと思われる。

計算機クラスタにおいては、ワークステーション、パーソナルコンピュータの機種や性能が異なる場合がある。また、複数のプロセッサによって構成された計算機クラスタは、多くの場合、個人所有ではなく複数の人々がハードウェア資源を共有するマルチユーザ環境である。マルチユーザ環境では、複数のユーザが自由に計算機を使用することから、計算機クラスタを構成する個々のプロセッサの負荷が動的に変動する。このような分散環境では負荷の重いプロセッサが処理の進み具合を遅らせ、結果として全体の処理が遅くなるという問題が発生する。そのために様々なスケジューリングの研究がなされている[2][3][4][5][6][7][8]。その中には、単一の並列・分散プログラムを対象としたロードバランシング方式も存在する[4][5][6]。

しかし、単一の並列・分散プログラムを対象とし、プログラム実行中に負荷の変動に伴った負荷分散を行なう方式はない。そのような実行方式を行なうためには、ユーザは自らロードバランシングを行なう必要がある。しかし、実際にそのような方式に並列・分散プログラムを書き換えるユーザの手間は複雑である。

実際に負荷分散をする際、ユーザにとってはアプリケーションプログラムの改変は最小限にとどめ、かつ可搬性を保つことが望ましい。また異なるアプリケーションに対して同じ負荷分散方法を適用し、負荷分散で行なうスケジューリングプログラム自体の再利用性を高めることができる。

本論文では、並列・分散アプリケーションプログラムにライブラリを統合して負荷分散を達成する方式を提案する。できるだけアプリケーションから独立し、アプリケーションに対して容易に追加でき、かつ効率の良い負荷分散を可能にする動的ロードバランサを実現する。

2 動的ロードバランサ pDLB の提案

2.1 pDLB の使用法

並列・分散アプリケーションプログラムに対して最低限の定義を行ない、ライブラリを統合して実行することによって、分散環境において動的負荷分散を行ないつつ、プログラムを実行する方式を提案する。本方式をライブラリとして実現したものを動的ロードバランサ pDLB と呼ぶ。

通常の並列・分散アプリケーション、例えばデータベース検索や離散事象シミュレーションなどを並列実行する場合、OpenMP[11]などを用い、アプリケーションを並列化したプログラムを実行する。それに対して pDLB を用いた並列・分散処理では、並列・分散アプリケーションプログラムに pDLB とのインタフェースをとる関数を組み入れ、そしてライブラリとして提供する pDLB と合わせてコンパイル、実行する。この実行過程で、動的ロードバランサ pDLB はアプリケーションの動的負荷分散を行なう。負荷分散のために pDLB が使用する通信においては、メッセージ通信インタフェース MPI[12]などを使用する。

2.2 対象とする並列・分散アプリケーション

動的ロードバランサを統合する対象となる並列・分散アプリケーションとしては、主に数値計算やデータベース検索、交通シミュレーション、流体シミュレーション、離散事象シミュレーションなどが考えられる。

このような計算では、関数に引数となる変数を渡して大規模計算を行なう。計算を行なう関数は同一であり、プロセッサ毎に渡される変数が異なることによって、独立して並列計算を実行できる。

本論文では、関数にある変数を渡すことによって実行される計算の単位をタスクと定義する。例えば、データベース検索アプリケーションの場合、1ファイルを対象に、1キーワードの検索を行なうことを1タスクと定義する。各プロセッサはタスクを多数持ち、その処理を行なう。このとき多数ある実行可能なタスクはプロセッサ毎にキュー（タスクキュー）に置かれ、プロセッサはキューからタスクを随時取り出し処理を行なう。本論文で提案、実装を行なう動的ロードバランサpDLBはこのような形式で並列処理を行なうアプリケーションを対象とする。

2.3 pDLBの負荷分散方式

データベース検索を例に動的ロードバランサpDLBの負荷分散の実行過程を、以下に示す（図1）。

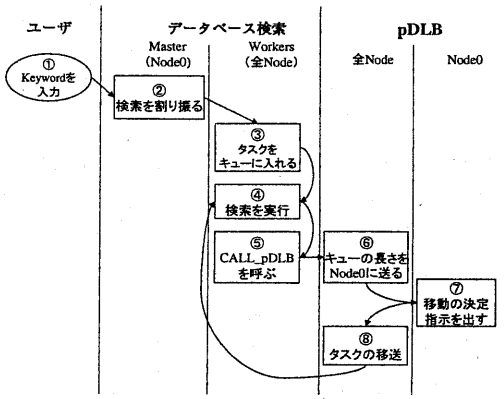

図 1 負荷分散の実行過程

① ユーザはNode0上で検索アプリケーションプログラムを実行し、Node0上でキーワードを入力する。

② Node0は並列プログラムの記述に従い、検索処理を全Nodeに割り振る。

③ 各Nodeは1ファイルに対して、1キーワードの検索を行なうことを1タスクとして、Node毎のキューに入れる。

④ 各Nodeはキューからタスクを順次取り出し検索処理を行なう。

⑤ Node毎に1タスクの処理を終える度にpDLBの関数CALL_pDLBを呼ぶ。

⑥ CALL_pDLBはキューの残りタスクの数をマスタNode0に送る。

⑦ Node0は残りタスクの数を比較して、タスク移送の可否を決定し、移送の指示を出す。

⑧ ワーカNodeは、指示に従ってキューかタスクを出し、送る。あるいは、タスクを受け取るNodeであれば、受け取ったタスクをキューに入れ、検索に戻る。

pDLBは、タスク移送をプログラム実行中に行なうことにより、動的に負荷を分散し均等化することを可能にしている。pDLBがタスクの移送により負荷を均一化する際、タスクの中身には関知せず、タスクという単位で移送をして負荷分散を行なう。従ってアプリケーションに依存しない形態となる。

3 動的ロードバランサpDLBの実装

アプリケーションプログラム上へのpDLBの統合の方法をデータベース検索アプリケーションの擬似コードを用いて示す（図2）。初めに、1ファイル、1キーワードを持つ構造体を定義する（1行目）。次にタスクを要素に持つ配列を定義し、タスクキューとする（2行目）。アプリケーションのメイン関数はタスクキューにタスクがなくなるまで処理を繰り返す（3、4行目）。計算処理を行なうためタスクを取り出す（5行目）。タスクより検索に必要なキーワード、ファイル名を取り出す（6行目）。変数を用いて検索を行なう（7行目）。プログラムに負荷分散を行なうために付加された関数CALL_pDLBを呼ぶ（8行目）。

```
/*データベース検索プログラム*/
1   typedef struct task{char keyword[256]; int filenumber;}TASK;
2   TASK *taskqueue[MAX_TASK];
3   main(){
4   while (!queue_empty(taskqueue)) {

5       TASK* tsk=gettask(taskqueue);
6       key=getkey(tsk); db=getdb(tsk);      /*keyword検索*/
7       search(key,db);

        /*動的ロードバランサ関数*/
8       CALL_pDLB_();
9       }
10  }
```

図 2　データベース検索アプリケーションの擬似コード

CALL_pDLB 関数プログラムは図 3 の擬似コードで示される。関数 tasksize を用いて Node0 にタスクキューの長さを送る（3 行目）。もし Node0 であれば、全 Node のタスクキューの長さを比較し、負荷分散を行なうか否かを決定する。もしタスクキューの長さに負荷分散が必要な程度の差があれば、移送するタスクを決定し、移送対象の Node に指示を出す（4, 5, 6 行目）。

指示を出された Node がタスクの送り手 Node の場合には、タスクの受け手である destNode に taskget という関数を用いてタスクを送る（7, 8, 9 行目）。また受け手は送られてきたタスクを受け取り次第、taskput という関数を用いて、自分のタスクキューに入れる（10, 11, 12 行目）。そして、再びアプリケーションに戻り処理を続行する。

```
  /*CALL_pDLBプログラム*/
1 extern TASK *taskqueue[MAX_TASK];
2   int CALL_pDLB(){
      /*負荷情報の伝達*/
3     n=tasksize(); (nをNode0に送る)
      /*タスク移送の決定*/
4     if(Node1){
5           m_i =decision(0,...,k);
            (Node iからmだけdest jにタスク移送を指示)
6     }
      /*タスクの移送*/
7     for(i=0;i<m;i++){
8           tsk=taskget();
              (tskをdest jに送る)
9     }
10    for(j=0;j<n;j++){
11          taskput(tsk);
              (タスクを受け取ってtskに代入)
12    }
13  }
```

n=tasksize();
tsk=taskget();
taskput(tsk);
のみが
アプリケーション
から提供

図 3　CALL_pDLB の擬似コード

このようにライブラリを用い、プログラムに関数を組み入れることによって負荷分散を行なう。pDLB が作動していれば、負荷分散を行ないつつ、並列・分散アプリケーションプログラムが実行される。また、作動していなければ、負荷分散の行なわない状態で通常の並列・分散アプリケーションプログラムとして実行を行なうことが可能である。このような方式をとることによって、アプリケーションプログラムの変更を最小限にとどめ、可搬性を保つことを可能としている。また異なるアプリケーションに対しても同じ負荷分散方式を適用し、動的ロードバランサの再利用性を高めるということも可能にしている。

4　動的ロードバランサ pDLB の評価

4.1　データベース検索に対する評価

LAN を介した WS(Workstation)クラスタの環境において、pDLB を統合したデータベース検索アプリケーションを実行する。データベース検索に対し、pDLB を 統合した場合と統合しない場合について比較実験を行なった。検索するファイルは WS クラスタ内に置かれ、NFS(Network File System)で共有されている。実験では、3 台の性能の異なる WS を用いた。3 台の WS の性能は次の通りである。

- SUN Enterprise [248MHz×4],
- SUN Ultra10 [440MHz],
- SUN SparkStation5 [70MHz]

検索するデータファイル（平均 3 MB）は 25 個あり、2 つのキーワードについて検索を行なう。タスク総数は初期段階で 50 個とし、これを開始時に Enterprise に 12、Ultra10 に 10、SS5 に 28 割り振った。

途中で追加された検索要求に対する負荷分散の動作を見るため、実験開始 60 秒後にタスクを 10 個追加した。pDLB を用いない場合は Enterprise に追加、用いた場合は SS5 に追加を行

なった。実行結果はpDLBを統合しない場合が図4、DLBの統合した場合が図5である。

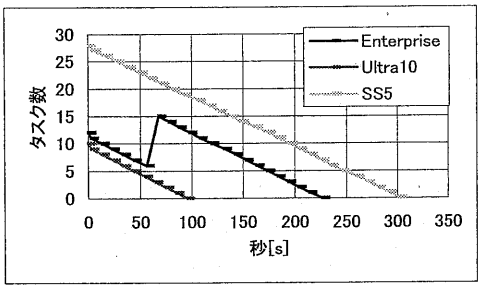

図 4 データベース検索:pDLBを統合しない場合

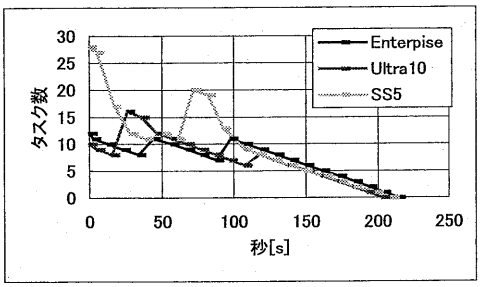

図 5 データベース検索:pDLBを統合した場合

実行結果を比較するとpDLBを用いない場合、WSによって実行終了時間が開始から100秒から300秒と開きがある。pDLBを用いた場合は、タスクの移送して負荷分散を行ない、その結果、各WSの実行終了時間が200秒周辺に集まっている。全検索処理の終了時間はpDLBを用いた方が速いことがわかる。またpDLBが初期段階のタスクに対応するだけでなく、60秒後に追加された検索要求に対しても負荷分散を行なっていることがわかる。このような結果から、pDLBが動的負荷分散により実行時間の短縮を達成することが検証され、その有効性が確認できた。

5 WaTorシミュレーションに対する評価

WaTorシミュレーション[9]は、生態系における生物の増減をシミュレートするプログラムである。このシミュレーションの目的は生態学的なリズムを再現することにある。このプログラム上では魚とそれを食べる鮫の2種類の生物を扱う。トーラス状になったセル空間内で魚が繁殖し、鮫が捕食、繁殖、死滅を繰り返す。プログラムを実行すると、生存する魚と鮫の数が時間とともに大きく振動する様子を描出できる。WaTorシミュレーションは発表された当初、逐次で実行されるプログラムであった、しかし現在では並列プログラムが作成されている[10]。pDLBの有効性を評価するために、[10]を参考にして逐次WaTorプログラムをOpenMP[11]で並列化した。このプログラムにpDLBを統合し、評価を行なった。

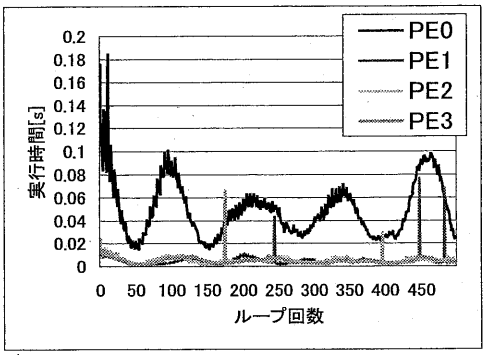

図 6 WaTorシミュレーション:pDLBを統合しない場合

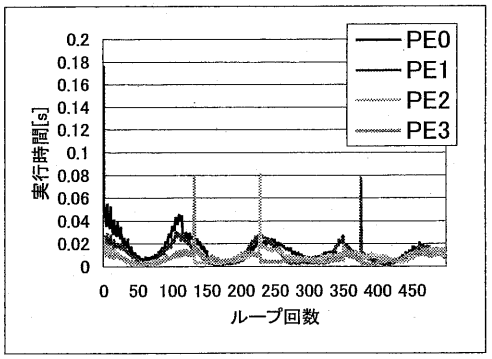

図 7 WaTorシミュレーション:pDLBを統合した場合

評価実験には、SMP(symmetric multiprocessor)計算機SUN Enterprise [248MHz×4]を用いた。初

期設定として、魚を1250匹、鮫を125匹とし、総ループ回数500回とした。50×50のセル空間に対して、PE0に全体の半分を割り振り、残りの半分をPE2,3,4で処理する。1ループ当りの実行時間の推移を図6、図7に示す。

pDLBなしの場合、PE0の実行時間は他のPEより大きい。それに対してpDLBありの場合、全体の負荷を均一化して全PEの処理時間を近い値にしている。500ループの実行を行なった結果、全実行時間はpDLBなしの場合は平均105.8[s]、ありの場合は平均89.8[s]になった。負荷分散の結果、全実行時間を短縮することが可能になった。

図6、図7において、実行時間にピークが数度表れるが、これはWaTorシミュレーションにおける固有のものである。SMP計算機を用いても、pDLBはWSクラスタの場合と同様に有効なタスク移送を行ない、負荷分散を達成できている。pDLBは様々なハードウェア環境で実装可能である。

6 おわりに

本論文では、データベース検索および、WaTorシミュレーションに対してpDLBの統合を行ない、有効性の確認をした。その際アプリケーションプログラムの改変は、プログラムにpDLB用の関数を付加するだけである。ライブラリ化によって、タスクの中身には関知しない形式でpDLBは動的負荷分散を達成した。種々のアプリケーションに対して容易に統合可能である動的ロードバランサを実現した。

今後の課題としては、pDLBの有効性を確かめるために、さらなるアプリケーションへの統合実験を行なうことがあげられる。また、動的ロードバランサpDLBのタスク移送を行なうためのより良い通信方式の検討が必要である。メッセージ通信インタフェースMPI[12]やPVM,JAVAのRMI等の導入が考えられる。

参考文献

[1] Myrinet .Myricom, http://www.myri.com/

[2] Behrooz A.Shirazi, Ali R.Hurson, Krishna M. Kavi, "Scheduling and Load Balancing in Parallel and Distributed Systems," the IEEE Computer Society Press, pp.1-4

[3] Load Sharing Facility .LSF, Platform Computing Corporation, http://www.platform.com/

[4] Berman.F, Wolski.R, Schopf.S.FJ, Shao,G, "Application - Level Scheduling on Distributed Heterogeneous Networks, " Proceedings of Super Computing `96,1996.

[5] Jon B.Weissman,Xin Zhao,"Scheduling Parallel Applications in Distributed Networks, " Journal of Cluster Computing Vol 1(1), 1998.

[6] Jon B.Weissman, Xin Zhao, "Run-time Support for Scheduling Parallel Applications in Heterogeneous NOWs," Proceedings of Sixth International Symposium on High Performance Distributed Computing (HPDC-6), August 1997.

[7] 中田秀基,竹房あつ子,松岡聡,佐藤三久,関口智嗣,"グローバルコンピューティングのためのスケジューリングフレームワーク,"並列処理シンポジウム論文集(JSPP`99), pp.227-284,1999

[8] LaxmikantV.Kale, SanjeevKrishnan,"CHARM++: A Portable Concurrent Object Oriented Sysytem Based On C++," Department of Computer Science, University of Illinois at Urbana-Champaign.

[9] " コンピュータレクリエーションI," A.K.デュードニー, 別冊日経サイエンス82, pp.71-78, 日経サイエンス社, 1987.

[10]Ladislav Hluchy, Vladimir Chovanec,"WaTor-An Irregular Distributed Simulation Problem,"MOSIS`98, pp.229-235,1998

[11] OpenMp: Simple, Portable, Scalable SMP Programming http://www.openmp.org/

[12] MPI: A Message-Passing Interface Standard, Massage Passing Interface Forum, June 12,1995.