2G−2

分散処理環境に適したプロセス・サーバと並列シェルの実現

片上　英樹、新城　靖、清木 康
筑波大学

1．はじめに

高性能で安価なマイクロプロセッサとＬＡＮの発展により、高性能ワークステーションがネットワークで結合された計算機環境が普及してきた。現在我々は、こうした並列・分散処理環境を有効に利用するための分散型オペレーティング・システムＲｅＳＣの実現を進めている[1]。本システムが対象とする並列・分散処理環境は、サイト間が高速ネットワークで結合されたものである。ここでサイトとは、シングルプロセッサやバス共有型マルチプロセッサをさす。

本稿では、本システムのプロセス・サーバと並列シェルの機能、およびＵＮＩＸ上でのプロトタイプの実現方法について述べる。

2．並列シェルの目的と全体構成

並列シェルの目的は、コマンド群を複数のプロセッサ上で並列に実行することによって、処理時間を最小にすることである。そのために、ネットワーク上のプロセッサの負荷情報とコマンドの結合関係を利用したプロセッサ割り当ての最適化を行う。

システムの全体構成を図1に示す。並列シェルは、構文解析モジュール、分散オプティマイザ、および実行モジュールから構成される。構文解析モジュールは、シェル・スクリプトを解析する。分散オプティマイザは、並列・分散環境を観察し、資源割り当ての最適化を行う機能を持つモジュールであり、コマンドを実行するサイトを決定する[1]。実行モジュールは、プロセス・サーバにプロセス生成要求を出し、プロセスの終了を管理する。

例えば、次のようなシェル・スクリプト

```
com-1 | com-2 | com-3
```

が与えられると、並列シェルは、com-1、com-2、com-3 の３つのコマンドの実行サイトを決定し、その通信路を設定して、プロセス・サーバを呼出し、プロセスを生成させる。並列シェルは、独立に実行可能なコマンド群を複数のプロセッサ上で並列に実行させることもできる。

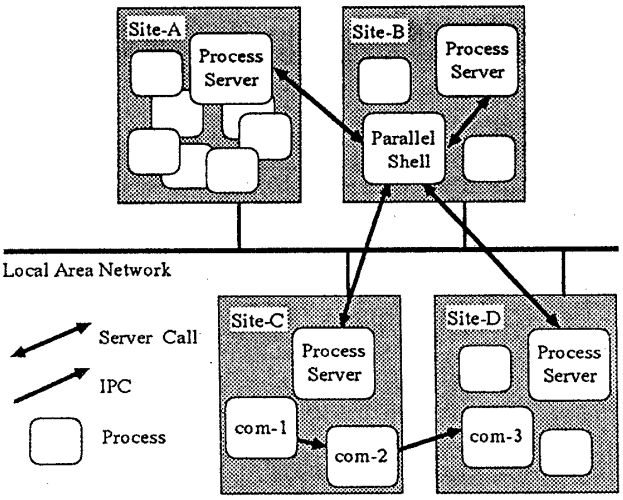

図1　並列シェルの全体構成

プロセス・サーバは、並列シェルや他のアプリケーションからの要求に従い、局所サイト上にプロセスを生成する。生成されたプロセスは、遠隔プロセッサ上のプロセスとも通信することができる。

3．プロセス・サーバの機能

プロセス・サーバは、次のような機能を提供する。

・プロセスを生成する。
・プロセスの終了を待つ。
・プロセスを強制終了させる。
・負荷情報を提供する。

これらの機能は、それぞれ、次の４つのサーバ・コールに対応する。

・pid　　Process_Create(program, arg)
・status　Process_Wait(pid)
・void　　Process_Terminate(pid)
・load　　Get_Load()

program は、実行するプログラムのロード・モジュールを指定する。arg は、文字列引数、環境変数、通信ポート、ファイルのケーパビリティなど、プログラムに与える引数を含む。pid は、プロセス識別子である。 status は、プロセスの終了状況などの情報を含む。 load は、サイトの負荷に関する情報を含む。

4．並列シェル

4．1　構文

並列シェルの構文は、ＵＮＩＸのシェルの構文に準じている。パイプラインは、“|”（縦棒）で区切られた複数のコマンド列から構成される。

＆（アンパサンド）で結合された文の解釈がＵＮＩＸのシェル・スクリプトと異なり、全てのコマンドの実行が終了した後、次の文の解釈・実行に移ることを表す。これは、次の例のように、並列処理の結果が後で利用されることを想定したものである。

```
cc -c a.c & cc -c b.c & cc -c c.c & cc -c d.c
ld a.o b.o c.o d.o /lib/libc.a
```

4．2　プロセッサ割り当ての方法

並列シェルは、コマンドを実行するプロセッサを決定するために、各サイトのプロセッサの負荷情報とシェル・スクリプトのコマンド数、およびコマンドの結合関係を利用する。ここでは、次のパラメタを使ってプロセッサ割り当ての方法を説明する。

n　：　コマンド数
a　：　利用可能なプロセッサ数
m　：　各プロセッサに最適な多重度
Ci　：　ｉ番目のコマンド　（０≦ｉ≦ｎ−１）
Pi　：　ｉ番目のコマンドを実行するプロセッサの番号
　　（０≦Ｐｉ≦ａ−１）

プロセッサ割り当てとは、Ｐｉを決定することである。以下に説明する割り当て方法では、次のことを仮定している。

・各コマンドの処理の重さがほぼ同じであること。

Implementation of a Process Server and Parallel Shell for Distributed Processing Environments

Hideki KATAGAMI, Yasushi SHINJO and Yasushi KIYOKI
University of Tsukuba

・各プロセッサの処理能力が同じであること。
・各プロセッサのプロセス生成コストは同じであること。
・ネットワーク通信が十分速いこと。

“&”結合のプロセッサ割り当て

次のようなコマンド列について、プロセッサ割り当てを行う。
C0 & C1 & … & Ci & … & Cn-1 （iは0…n－1）
“&”で結合されたコマンド群は、コマンド間で通信を行わないので、プロセス間通信のコストを考えずに実行させることができる。このようなコマンド群に対するプロセッサ割り当ての方法を示す。

◎ “&”結合の割り当て方法

n≦m×aのとき
Pi ＝ i mod a

n＞m×aのとき

step-1) 0からm×a－1までの整数iについて
Pi ＝ i mod a

step-2) Piのプロセッサで、コマンドの実行が終了するたびに、残りのコマンドを1つずつ分配する。

これは、プロセッサの処理状況を観察しながら、徐々にプロセッサを割り当てる方法である。その利点は、負荷の変動に応じた割り当てが可能であることである。

“|”結合のプロセッサ割り当て

次のようなコマンド列について、プロセッサ割り当てを行う。
C0 | C1 | … | Ci | … | Cn-1 （iは0…n－1）
“|”で結合されたコマンド群に対する最適なプロセッサ割り当てにおいて、プロセス間の通信コストを考えることが重要となる。ネットワーク通信のコストが大きい場合、並列処理を行うよりも、1つのサイトを使用した方が効果的である可能性がある。ネットワーク通信がボトルネックにならないハードウェア環境では、通信によるコストが大きくなっても並列に処理するほうがコマンド群の処理時間が小さくなる。そこで、並列処理を優先した割り当て方法をとる。

次に、2つの割り当て方法を示す。

◎ “|”結合の割り当て方法1
（通信コストを減らす）

n＜aのとき
Pi ＝ i

n≧aのとき
Pi ＝ i div （n div a）

方法1は、通信コストを小さくすることを目的としたものである。連続して結合された幾つかのプロセスを同一サイトで実行することによってネットワーク間の通信を減らし、通信コストを小さくしている。ただし、パイプラインの非定常状態では、遊んでいるサイトが生じる。

◎ “|”結合の割り当て方法2
（パイプライン処理における負荷を均衡させる）

Pi ＝ i mod a

方法2は、パイプライン処理における負荷の均衡を目的としたものである。各プロセッサに、コマンドを1個づつ、順番に割り当てることによって、パイプラインの非定常状態の負荷の均衡を図る。ただし、この方法では、ネットワーク通信のコストが大きくなる。

5． 並列シェルとプロセス・サーバのＵＮＩＸ上での実現

並列シェルとプロセス・サーバのプロトタイプをＵＮＩＸ上に実現した。並列シェルとプロセス・サーバ間の通信は、ＳｕｎＲＰＣを使用した[2]。プロセスの生成には fork() システムコールを使用した。コマンド間のプロセス間通信はＴＣＰ／ＩＰのストリームを使用した。入力側が connect() システムコール、出力側が accept() システムコールを発行する。ユーザとの対話を実現するため、ターミナル・サーバを作成した。ターミナル・サーバとプロセスの間は、ＴＣＰ／ＩＰのストリームで結合される。

6． 実験

パイプラインを構成するコマンド群を並列シェルで実行させる実験を、イーサネットで結合された3台のワークステーション（Ｓｕｎ４）上で行った。コマンド数を9（ｎ＝9）、利用可能プロセッサ数を3（ａ＝3）とした。コマンド間の通信量とコマンドの負荷を変えた2種類のスクリプトを用いて、処理時間を比較した。2種類のスクリプトは、次の性質がある。

・通信量大・負荷小
（プロセス間通信量 ： 1Ｍ bytes
（1Ｋバイト当たりの処理時間 ： 0 sec
・通信量小・負荷大
（プロセス間通信量 ： １０Ｋ bytes
（1Ｋバイト当たりの処理時間 ： １６０ msec

前者のスクリプトでは、“|”結合の割り当て方法1が有効であると予想される。逆に、後者では、“|”結合の割り当て方法2が有効であると予想される。

実験結果を表1に示す。

表1 各割り当て法の処理時間の比較（単位：秒）

	並列処理		逐次処理	ＵＮＩＸパイプ
スクリプト	方法1	方法2		
通信大負荷小	5．7	8．5	１４．２	１２．０
通信小負荷大	7．7	7．0	１７．６	１６．７

割り当て方法を以下に示す。

・“|”結合の割り当て方法1による並列処理
・“|”結合の割り当て方法2による並列処理
・逐次処理
・ＵＮＩＸのパイプ

逐次処理と並列処理では、いずれのスクリプトでも後者の方が処理時間が小さい。予想されたとおり、スクリプト1では方法1が方法2に勝り、スクリプト2では逆になった。ＵＮＩＸのパイプと本並列シェル（ＴＣＰ／ＩＰ）による逐次処理とを比較すると、前者の方が処理時間が小さくなった。

7．終わりに

並列シェルとその構成法、プロセッサ割り当て法について述べた。また、プロセス・サーバの機能について述べた。今後の課題は、コマンドの処理の重さやプロセッサの処理能力が異なる場合にも最適となる割り当て方法を考え、そのシステムの実現と評価を行うことである。

参考文献

[1] 新城、清木、益田：“並列・分散処理環境を対象としたＯＳＲｅＳＣにおける分散オプティマイザ”、情報処理学会第４０回全国大会講演論文集、5G-3、pp.724-725（1990）

[2] "Network Programming Guide"、Sun Microsystems, Inc.（1990）