Vol.2015-OS-134 No.21
2015/8/5

Plan9 による分散シェルシステム

中原健志[†1]　佐藤美紀子[†1]　紫合治[†2]　並木美太郎[†1]

近年，分散処理の利用場面が増えつつあるが，多くの分散処理系では専用のプログラムを書く必要があり，プログラム変更の手間が大きい．本稿では既存のコマンドを分散して実行する分散シェルについて述べる．本シェルではデータの分割，合成とネットワークを介して利用可能なパイプの機能を追加し，コマンドをネットワークで繋がっている複数台の計算機で分散実行することでプログラムを改変せず手軽に分散処理を実現することを検討した．コマンドの分散実行については，各種の位置透過性が必要であるが，これらを分散向け OS である Plan9[1]の分散透過性を用いることでシェルに実現する．

Distributed Shell System based on Plan9

TAKESHI NAKAHARA[†1]　MIKIKO SATO[†1]
OSAMU SHIGO[†2]　NAMIKI MITARO[†1]

1. はじめに

近年，処理するデータ量の増大などに伴い，分散処理への関心が高まっている．分散処理を実現する手法としては Hadoop や MPI などが広く知られている．しかし，Hadoop や MPI などでは分散処理させたいプログラムをそれぞれの処理系が定義している命令などで書き換える必要がある．このため，利用者はそれら処理系の命令について習得する必要がある．また，プログラミング言語についてもそれらの処理系に対応している必要があり，利用者は自分の慣れ親しんでいる言語を利用できない場合がある．これらの問題点から利用者は気軽に分散処理を行うことが難しい．そこで本稿では UNIX の基本概念である「パイプとフィルタによる並列処理」に入力データの分割と出力データの合成の機能を追加し，ユーザが指示したコマンドを複数のリモートの計算機で実行することにより，容易に分散処理を実現することを検討した．本機能を備える分散シェルを，分散向け OS である Plan9 の持つ透過性の機能を用いることで少ない実装で実現した．

2. 関連研究

2.1 GNU Parallel2[1], DSH[3] PDSH[4]

GNU Parallel は，複数台のリモートの計算機へ SSH を用いて接続し，指定されたコマンドを実行することが出来る．しかし，リモートの計算機上にコマンドのプログラム，または処理に必要なファイルがない場合には，自分で scp コマンド，もしくはオプションで処理に必要なファイルを指定して転送する必要がある．このため，利用者はリモートの計算機上に必要なファイルがあるかを常に意識する必要がある．

2.2 GXP[5]

GXP では，ファイルやディレクトリの差異を GMount[6]を用いてそれぞれのリモートの計算機の特定のディレクトリをつなぎ合わせて，一つの大きな共有ディレクトリに見せることで，解決している．しかし，UNIX 系 OS の名前空間の制限から，名前付きパイプなどの特殊なファイルは共有されない．

3. 分散 OS：Plan9

Plan9 は Bell 研究所によって開発された分散環境向け OS である．Plan9 は下記の４つのサービス（サーバ）からシステムが構成されている．

- **Auth サーバ：認証を行う**
- **CPU サーバ：計算処理専用のサーバ**
- **File サーバ：ファイル管理を行う**
- **Terminal：端末**

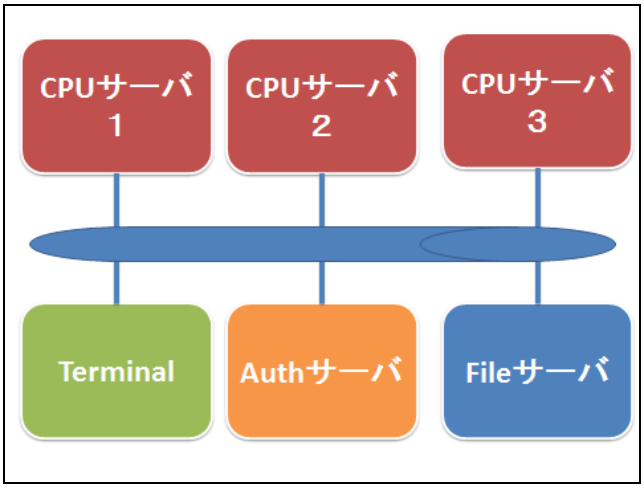

図 1：Plan 9 システム構成図

†1 東京農工大学
Tokyo University of Agriculture and Technology
†2 東京電機大学
Tokyo Denki University

これら４つのサービスは１台の計算機上でも，または，サービスごとに計算機が分かれ，ネットワークで接続されている形でも可能である．そして Plan9 は分散システムを実現するにあたり，次の２つの透過性を実現している．

(1) **アクセス透過性**

ネットワークを含むほとんどの計算機資源をファイルへ抽象化している．これにより，利用者は様々な計算機資源の違いを意識せず，ファイルへの読書きで利用できる．

(2) **位置透過性**

Plan9 ではプロセスごとに名前空間が独立している．これにより，あるプロセスで名前空間に変更を加えても，他のプロセスの名前空間には影響を与えない．この利点として意図しない利用者が自分の名前空間へアクセスすることを防いでいる．また，名前空間は自由に編成することが可能である．例えば，ネットワークを介してつながっている計算機のファイルを自分の名前空間へ取り込み，ローカルの名前空間と置き換えることも可能である．これにより計算機資源がローカルであっても，リモートであっても利用者は場所の違いを気にせずに利用することが可能である．

3.1 9front[7]

Plan9 は Bell 研究所での開発がほぼ終了した状況となっており，ドライバなどの更新が行われていない．そこで，9front と呼ばれる Plan9 から派生 OS の開発版を利用する．9front では有志によってドライバの更新，および機能拡張が行われている．Plan9 ではいくつかのハードウェアを認識しない問題があったため，本研究では 9front を利用して実装を行っている．

4. 本研究の目標

今日，広く利用されている Hadoop や MPI などの分散処理系では，プログラムに対して大幅な変更を加えることで分散処理を実現している．しかし，プログラムへの変更が必要なため，これらの処理系は気軽に利用することが出来ない．また，HTCondor[8]などのバッチスケジューラによる分散処理系ではバッチジョブを記述する必要があるが，設定項目が多く，処理の流れが直観的ではない．そこで，処理させたい内容を粗粒度な単位で細分化し，それぞれを一つの入力と出力を持つコマンドとして実装する．それらをパイプで繋ぐことで手軽に簡易的な分散処理を実現することを検討した．この特徴を持つ処理系に GNU Parallel, DSH, PDSH，GXP などが挙げられる．しかし，Linux では位置透過性の機能が欠けているため，利用者はローカルとリモートの計算機でのファイルやディレクトリの違いに注意する必要がある．

本研究では，処理の記述方法に UNIX のパイプの記号を発展させた記号を用いて，分散処理させたいコマンドを指定する．

分散処理されるコマンドは，リモートの計算機上で実行される．この際，リモートの計算機の名前空間をローカルの計算機の持つ名前空間で置き換えることで，ファイルやディレクトリの差異を減らすことを検討した．これにより，利用者はファイルやディレクトリについて，リモート・ローカルの違いを気にせず，利用可能にする．

これらに加え，近年普及しつつある CPU 以外の FPGA や GPU といった異なるアーキテクチャーのプロセッサをファイルへ抽象化する．さらに，これらの入出力をコマンドへ抽象化させることで，ヘテロジニアスな環境を利用可能なシェルの開発を目指す．

上記の内容を実現する際に，通常の UNIX 系の OS では透過性に欠けているため，実現が難しい．そこで本研究では位置透過性とアクセス透過性の機能を持つ OS : 9front を用いた．

本稿では現時点での進捗状況について説明する．

5. システム構成

本研究で開発した分散シェルシステムはフロントエンドである分散シェル，分散シェルの記号の解釈・変換を行う parser とデータの分割と合成の機能から成り立つ．利用者は初めに, 利用する CPU サーバのアドレスを分散シェルへ登録する．その後，分散シェル記号（**||, [,]**）を用いて実行させたいコマンドを記述する．コマンド列の記述が完了すると分散シェルは記述内容を parser へ渡す．parser は分散シェル記号で記述されたコマンド列を 9front のコマンド列へ変換していく．このとき，parser で必要な数の名前付きパイプの生成，それぞれのコマンドの標準入出力を名前付きパイプへ変更，名前空間の置き換え，データの分割と合成コマンドの追加を行う．コマンド列の変換が完了すると処理は 9front へ渡され，記述されたコマンド列が実際に実行される．

5.1 分散シェルシステムの設計方針

分散シェルシステムでは比較的大きな処理を粗粒度な単位に分割し，分割された粗粒度な処理に対して，入力と出力を一つずつ持つコマンドで実装する．そして，これらのコマンドをパイプで繋げることで大きな処理を行うことを想定している．この手法の利点として次の４つが挙げられる．

(1) **既に使われているコマンドを容易に再利用可能**

単純なコマンドとパイプを組み合わせて処理を行うため，様々な処理へコマンドを応用可能である．

(2) **言語の制限の少なさ**

プログラミング言語で標準入力と標準出力をサポー

トしていれば良いため，開発言語の制限が少ない．

(3) **プログラムの簡素化**

処理全体を粗粒度な単位で分け，各処理をコマンドとして実装する．コマンド列で大きな処理内容を記述するため，処理内容全体が把握しやすい．

(4) **分散処理による性能向上**

複数台の計算機へ処理を分散させるため，１台あたりの処理量が減り，性能を向上できる．

これらのコマンドを分散シェル記号により，どのコマンドを分散処理したいのか，そして分散する際のデータの分割と合成の方法を指定していく．分散シェル記号は通常の UNIX のパイプ記号"|" を基にした３つの記号（||, [,]）がある．この記法の利点として次の２点が挙げられる．

- UNIX のパイプの記号を分散シェル記号に置き換えるだけで分散処理が可能であり，習得が容易である
- 視覚的にどのコマンドが分散処理されるかが分かりやすい

本シェルシステムでは，9front の透過性の機能を用いて，リモートのサーバのホームディレクトリを利用者の端末側のホームディレクトリへ置き換えている．そのため，利用者はコマンドを分散実行させる場合であっても，ホームディレクトリ上にあるファイルやディレクトリを，実行場所や path の違いを気にせず利用可能である．

上記の特徴から本シェルシステムはプログラムの改変をすることなく，手軽に分散処理を実現できる．

5.2 分散シェルによる分散実行

分散シェルはフロントエンド部とコマンドを解釈する「parser」，そして parser が生成したコードを実際に実行する 9front のシェル「rc」から成り立つ．利用者は rc から分散シェルを立ち上げる．その後，利用可能な CPU サーバの IP アドレスを分散シェルへ登録する．
登録後，利用者は分散シェル記号を用いて，処理したいコマンド列を記述する．記述内容はフロントエンドから parser へ渡される．parser はコマンド列を解釈し，9front のシェル「rc」の文法へ変換，シェルスクリプトとして出力する．変換時に分散処理に必要になる中間ファイル（名前付きパイプ）の生成命令，およびそれぞれのコマンドの入出力を中間ファイルのものへ変更する．変換が完了後，分散シェルは，生成されたシェルスクリプトを実行し，コマンド列の処理が実際に実行される．

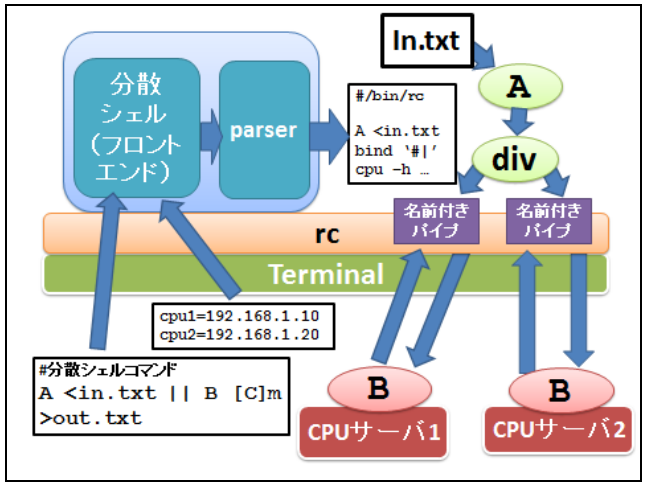

図 2　分散シェルによる分散実行

5.3 分散シェルの文法

分散シェルでは４つの記号（|, ||, [,]）を用いて，処理させたいコマンド列を記述していく．分散シェルの文法は次の通りである．

分散シェル文法:=分散コマンド[分散コマンド]…
分散コマンド:=
　　ローカル記号 | リモート記号 | 分散処理記号
ローカル記号:=
　　コマンド[〝|〞コマンド]…
リモート記号:=
　　〝||〞コマンド
分散処理記号:=
　　〝[〞 ローカル記号 { 〝]〞 | 〝]m〞 | 〝]r〞 }
コマンド:=
　　コマンド名[引数]… [〝<〞 入力] [〝>〞 出力]

表 1　分散シェルの文法

5.4 分散シェルの記号

- **|：ローカル記号**

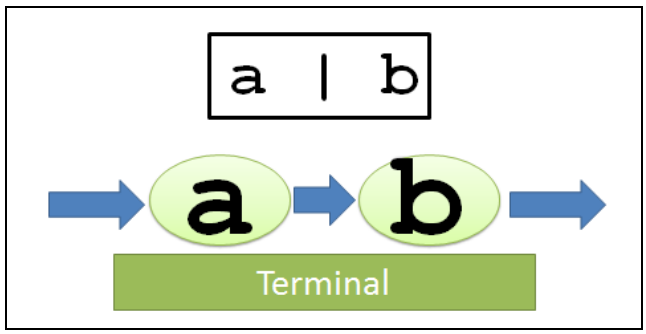

図 3　ローカル記号でのコマンド処理

UNIX のパイプ記号と同様に，ローカル記号の次に書かれたコマンドは利用者の端末上で実行される．分散シェル上でコマンドに対して記号の指定がない場合は，ローカル

記号があるものとして分散シェルは実行する．

- **||：リモート記号**

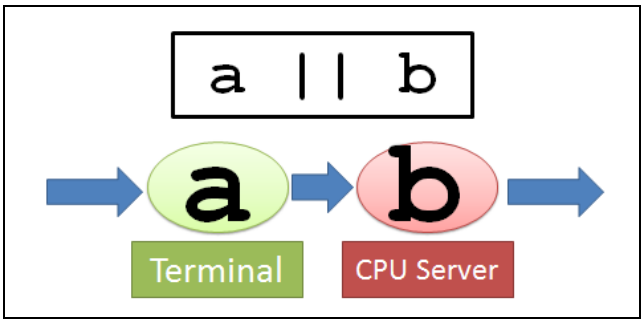

図 4　リモート記号でのコマンド処理

リモート記号の次に書かれたコマンドは，1 台の CPU サーバ上で実行される．この時，どの CPU サーバを使うかは分散シェルが自動的に決める．また，CPU サーバのホームディレクトリを，利用者の端末側のホームディレクトリへ置き換える．これにより端末と CPU サーバ間のファイルやディレクトリの差異を減らしている．

- **[コマンド]：分散記号**

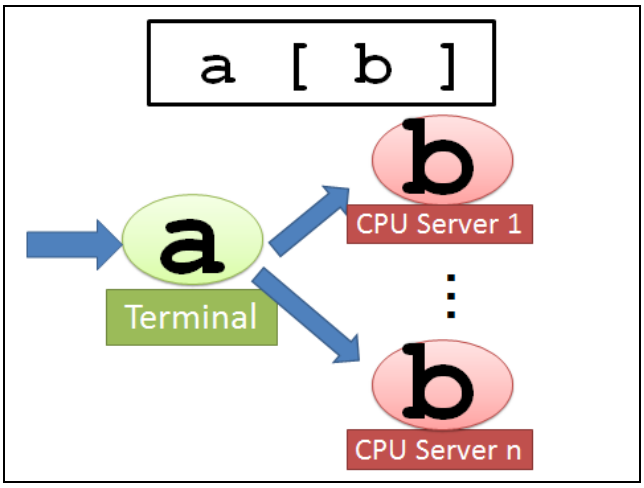

図 5　分散記号でのコマンド処理

分散記号で囲まれたコマンドは，利用可能なすべての CPU サーバ上で実行される．この時，分散シェルは，分散記号の前にあるコマンドからの標準出力を改行ごとに，それぞれの CPU サーバ上で実行されるコマンドの標準入力へ順番に渡す．それぞれの CPU サーバで処理され，標準出力へ出された結果は，再び分散シェルが一つにまとめ，分散記号の後ろにあるコマンドへ渡す．この処理結果のまとめ方は利用者が指定することができ，次の 3 つを提供している．

- **]m：ソートマージオプション**
  それぞれの CPU サーバでの処理結果を比較し，小さいものから出力していく．利用場面として，ソートを行う際に使われる．
- **]r：ラウンドロビンオプション**
  それぞれの CPU サーバからの出力を順番に出力する．利用場面として，表データといったデータの並びを維持する必要があるものへの処理に使われる．
- **]：デフォルト**
  CPU サーバから処理が出力された順にまとめ，出力していく．利用場面として，データの並びなどが重要でないものに使われる．

もし，オプションの指定がなかった場合，処理結果のまとめ方はデフォルトが設定される．

6. 分散シェルでの内部処理

分散シェルで記述されたコマンド列は parser へ渡される．parser は，分散シェル記号で記述されたコマンド列を 9front の標準シェル"rc"ものへ変換する．変換する際に主に 9front に標準で搭載されている次の 3 つのコマンドを利用している．

●CPU コマンド
指定された CPU サーバへ接続し，指定されたコマンドを実行する．この時，端末のルートディレクトリが CPU サーバの「/mnt/term」以下へマウントされる．

●名前付きパイプ
9front の名前付きパイプでは，指定されたディレクトリ以下に 2 つの仮想的なファイルを生成する．これら 2 つのファイルは等価なファイルであり，片方にデータを書き込むと，もう片方からも書き込んだ内容を読みだすことができる．また，9front の名前付きパイプはネットワークを介して，他の計算機から読書きすることが可能である．

●bind コマンド
名前空間の拡張・置き換えを行う．

6.1 コマンドの実行

分散シェル上で記述された各コマンドは次の手順で 9front のコマンドへ変換・実行される．

1. ローカル記号
ローカル記号はコマンドを端末上で実行する．もし，ローカル記号の前にコマンドが記述されていた場合は，そのコマンドの標準出力，または名前付きパイプをコマンドの標準入力へつなぐ．また，ローカル記号で指定されたコマンドの後ろにコマンドが記述されていた場合は，標準出力を名前付きパイプへ変更する．

```
➢  bind '#|' pipe1
➢  コマンド <in.txt >pipe1/data
```

図 6　ローカル記号の変換例（疑似コード）

2. リモート記号

リモート記号で指示されたコマンドは 1 台の CPU サーバ上で実行される．どの CPU サーバを利用するかはシェルが自動的に決定する．CPU サーバ接続時に，9front の bind コマンドを用いて，CPU サーバのホームディレクトリを端末のホームディレクトリへ置き換える．これにより，CPU サーバのホームディレクトリより下位のディレクトリでは，端末と等価なディレクトリ構成となっている．

```
➢  bind ‘#|’ pipe2
➢  cpu0=’192.168.11.10’
➢  cpu -h $cpu0 -c ‘
➢       bind -b -c /mnt/term/home $home; ’
➢       cd 端末の現ディレクトリ
➢       コマンド <pipe/data1 >pipe2/data
```

図 7　リモート記号の変換例（疑似コード）

3. 分散記号

分散記号で指定されたコマンドは，利用可能なすべての CPU サーバ上で分散実行される．分散実行する際には利用可能な CPU サーバ台数分のデータ入力用と出力用の名前付きパイプを生成する．そして分散処理記号の前にあるコマンドの標準出力を改行ごとに分割し，以前，生成したデータ入力用の名前付きパイプへ順番に出力する．それぞれの CPU サーバで実行されるコマンドは入力用の名前付きパイプからデータを入力する．処理結果は，それぞれのデータ出力用名前付きパイプへ出力される．シェルではデータ合成コマンドを起動させ，それぞれの名前付きパイプを監視し，利用者が指定した方法で処理結果をマージし，標準出力へ出力する．

```
➢  bind ‘#|’ pipe3_0;
➢  bind ‘#|’ pipe4_0;
➢  bind ‘#|’ pipe3_1;
➢  bind ‘#|’ pipe4_1;
➢  div ‘pipe3_0/data’ ‘pipe3_1/data’
   < pipe2/data1
➢  for(n=0; n<3; n++){
➢     cpu -h $cpun -c
➢        bind -b -c /mnt/term/home $home; ’
➢        cd 端末の現ディレクトリ
➢        コマンド <pipe3_0/data1 >pipe4_0/data
➢  }
➢  merge ‘pipe4_0/data1’ ‘pipe4_1/data1’ >
   out.txt
```

図 8　分散記号の変換例（疑似コード）

入出力の分割と合成は，分散シェルに付属する次のコマンドへ変換される．

- **データ分割：div**

div コマンドは標準入力からの内容を指定された各名前付きパイプへ改行ごとにラウンドロビンに出力していく．

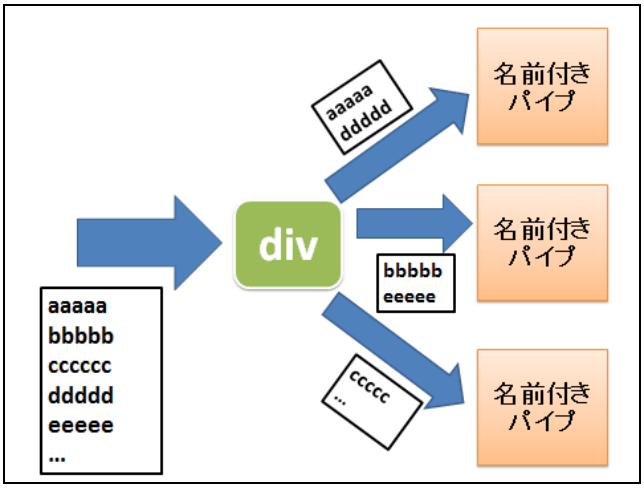

図 9　分散シェルでのデータ分割

- **データ合成**

それぞれの CPU サーバ上で動いているコマンドからの出力結果を，リモート記号のオプションに基づいて一つにまとめ，標準出力へ出力する．まとめ方には 3 つあり，それぞれ分散シェル記号の閉じ括弧の直後に文字を置くことで指定可能である．もし，オプション指定がない場合はデフォルトが指定される．

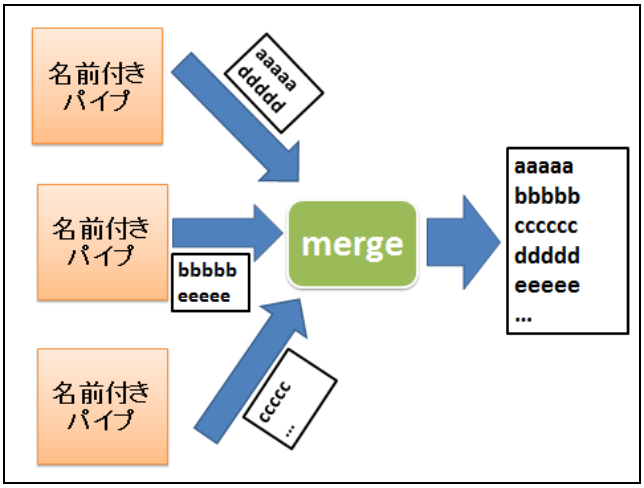

図 10　分散シェルでのデータ合成

- **マージオプション：]m**

マージオプションは，複数の名前付きパイプの内容を監視し，それぞれのデータを比較したうえで小さいものから出力していく．

- **ラウンドオプション：]r**
  ラウンドロビンオプションでは，複数の名前付きパイプの内容を改行ごとに順番に出力していく．
- **デフォルト：]**
  複数の名前付きパイプを監視し，内容があったものから出力していく．

データの分割と合成の２つの機能をシェルへ追加することで標準入出力を一つずつ持つコマンドでの分散処理を可能にした．

7. 性能評価

7.1 パイプの性能

今回，端末内とネットワークを介した場合の 9front のパイプ性能を測定した．測定環境として CPU：Core i7-2600（3.4Ghz），メモリー：8GB，OS：9front（2015/03/02 リリース版）をインストールしたものを２台用意した．それぞれの計算機は 1Gbps のイーサネットで接続されており，MTU は 1500Byte の設定である．

性能評価方法として，9front 上で名前付きパイプを生成し，２つのプロセスから名前付きパイプへの読書きを行った．片方のプロセスでは指定したバイト数のデータを，パイプへ書き込む．もう片方のプロセスでは，指定したバイト数のデータを読み込む．この作業を読み込み（書き込み）の総データ量が 16MB になるまで繰り返し，完了するまでの時間を計測した．結果は表２の通りである．

送受信サイズ(Byte)	端末内（msec)	CPU-端末（msec）
500	78	2591
1000	46	1964
2000	23	1246
4000	13	1031
8000	9	1027

表 2　16MB の読書き処理時間

この結果より，一回の転送データサイズが転送時間に影響することが分かった．また送受信サイズが 4000Byte 以上の場合，処理時間がおよそ１秒を超える処理では，分散処理におけるデータの転送オーバヘッドを補うことが可能であると考えられる．

7.2 単語の出現頻度計測による分散シェルシステムの性能評価

今回，単語の出現頻度を求めるプログラムを用いて，分散シェルシステムの性能評価を行った．
単語の出現頻度を求める手順は下記の通りである．

(1) **テキストを単語に分解**
(2) **単語でソート**
(3) **単語の出現回数を数える**
(4) **出現回数でソート**

今回，上記の処理を下記のコマンドを用いて行った．

- **word:**入力を単語に分解して出力
- **count:**単語で同じ単語を数えて「出現回数　単語」の形で出力
- **sort：**入力テキストをソートして出力

単語の出現頻度を求める処理中で最も重い処理は一回目の「ソート」処理である．よって，一回目の sort コマンドを複数台の計算機で分散実行させた．
分散シェルでは下記のように記述される．

- **sort(1)を複数台の CPU サーバで分散処理する**

```
word <in.txt [ sort(1) ]m count | sort(2) >out.txt
```

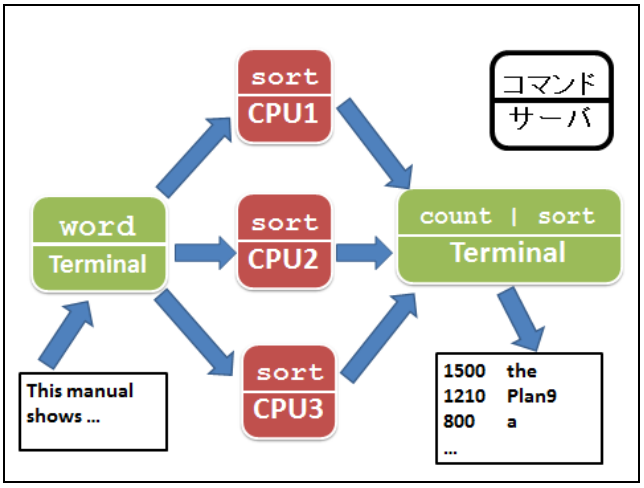

図 11　単語の出現頻度を求める処理の流れ

7.2.1 性能評価

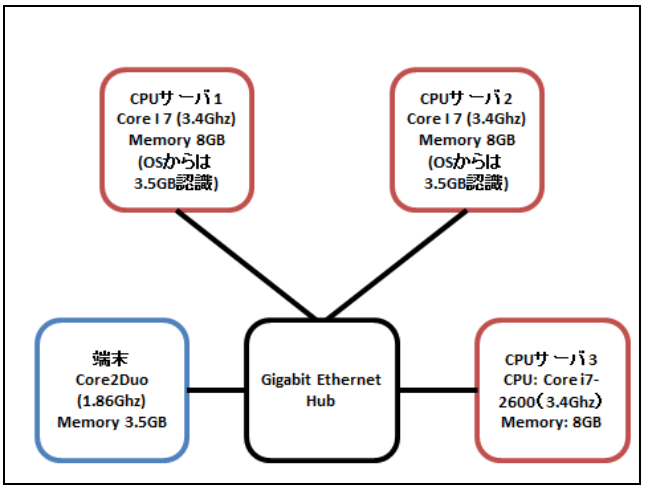

図 12　サーバ構成図

単語の出現頻度を求めるため，端末と３台の CPU サーバを用いて性能評価を行った．端末の性能は CPU：Core 2 Duo 1.86Ghz，メモリ:2GB，CPU サーバの性能は CPU:：Core i7-2600（3.4Ghz），メモリ：8GB であり，それぞれ 1Gbps のイーサネットで接続されている．

今回の測定では，ファイル入出力による性能評価への影響を減らすため，結果の出力を「/dev/null」へとした．また，巨大なファイルのソートでは，「/tmp」ディレクトリへ一時ファイルを生成する．そのため，「/tmp」をメモリ上で読み書きする（ramfs）様に設定した．

実際に処理に利用したコマンドは下記の２つである．

- **端末単体**

```
word <in.txt | sort | count | sort >/dev/null
```

- **分散処理**

```
word <in.txt [ sort ]m count | sort >/dev/null
```

7.2.2 測定結果

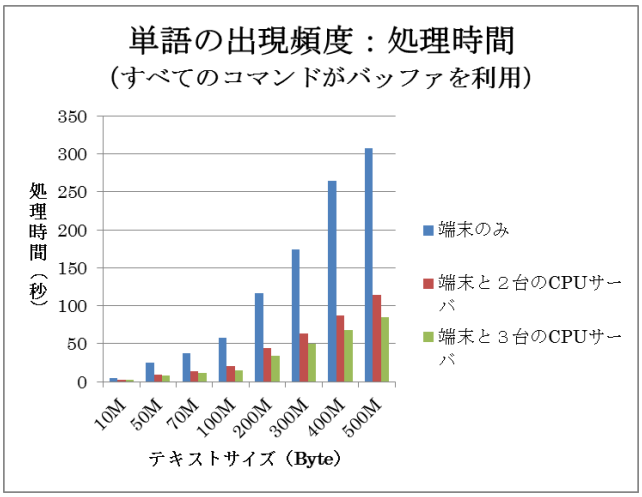

図 13　単語の出現頻度に要した処理時間

図１３の測定結果から，分散処理での処理時間が端末単体のものよりも短くなることが確認できた．CPU サーバ２台と３台で大きな違いがないのは通信によるオーバヘッドのためだと考えられる．

7.3 単語のヒストグラムによる評価

文章中で使われている単語の中で，指定した辞書ファイルに登録されている単語を探し，出現頻度順に表示するコマンド群を用いて，位置透過性の評価を行った．処理環境は単語の出現頻度を求めたものと同じである．

処理の流れは

(1) **テキストを単語に分解**
(2) **単語でソート**
(3) **辞書中に単語があるかを調べ，合致した単語のみ出力**
(4) **単語の出現回数を数える．**
(5) **出現回数でソート**

となる．

ここで(1)，(2)，(4)，(5)の処理は，単語の出現頻度を求める際に用いた，「word」，「sort」，「count」コマンドを用いる．処理(3)の辞書中に単語があるかを調べるコマンドは，9front に標準でついている「look」コマンドを用いる．look コマンドのオプションにファイルを指定した場合，そのファイル中にある単語と標準入力から入力される単語が一致するかを調べ，一致した場合のみ出力する．今回，9front の位置透過性を確認するため，辞書ファイルを端末のホームディレクトリに置いた．それぞれの CPU サーバのホームディレクトリは接続時に端末のホームディレクトリへ分散シェルが置き換える．そのため，利用者はホームディレクトリ上のファイル CPU サーバでも端末と同じように利用可能である．評価として，端末単体と CPU サーバ３台と 10MB のテキストを用いて，処理時間を計測した．利用したコマンド列は次のものである．

- **端末単体**

```
word <10M.txt | look -i -x -f words | sort |
count | sort >/dev/null
```

- **分散処理（CPU サーバ３台）**

```
word <10M.txt [ look -i -x -f words | sort ]m
count | sort >/dev/null
```

9front の位置透過性の機能により，端末単体と分散処理での look コマンドのファイルのディレクトリは全く変更する必要がない．このことが本シェルにおける特徴である．測定結果を表３に示す．

端末のみ	CPU サーバ(辞書はリモート)	CPU サーバ(辞書はローカル)
119 秒	1969 秒	82 秒

表 3　ヒストグラム作成の処理時間

処理結果から，CPU サーバによる分散処理が端末単体による処理よりも長く時間を要した．原因として，look コマンドが標準入力からの単語と辞書中の単語を比較する際に，CPU サーバのホームディレクトリ上にある辞書ファイルへアクセスする．しかし，実際には位置透過性の機能により，ネットワークを介して端末のホームディレクトリへアクセスされる．この時，ファイルからの読み込むデータサイズが小さいために，データの転送のオーバヘッドが大きくなったのではないかと考えられる．

逆に辞書ファイルを CPU サーバのローカルに置いた場合の分散処理では，端末よりも処理時間が短い結果となった．

8. おわりに

本稿では，UNIX のパイプとフィルタの概念を分散処理へ拡張し，データの分割と合成の機能を追加することで，入出力を一つずつ持つ単純なコマンドによる分散処理を実現した．また，9front の持つ透過性の機能が分散シェルを実現するのに適していることを示した．課題として，look コマンドで処理速度が大幅に低下した原因の調査とシェルシステムの改良を行う．現在の分散シェルでは名前付きパイプが端末上に生成されるため，CPU サーバの接続台数が増えるにつれ，端末への負荷が上がる可能性がある．また，複数のファイルからの入力，及び複数パラメータへの対応が改良すべき点として挙げられる．それらが完了後，9front のアクセス透過性の機能を用いて，FPGA などの異種プロセッサをコマンドへ抽象化し，シェルの内部で利用が可能かについて検討する．

参考文献

1) Rob Pike and Dave Presotto and Ken Thompson and Howard Trickey, Plan 9 from Bell Labs, In Proceedings of the Summer 1990 UKUUG Conference 1—9 (1990)
2) Ole Tange, “GNU Parallel: The Command-Line Power Tool”, February 2011, Volume 36, Number 1
3) Yoshiki Taniguchi, Junichi Uekawa, “Evaluation of time used in dsh in DNAS application start up sequence” http://mikilab.doshisha.ac.jp/dia/research/person/yoshiki/mlm52.pdf
4) PDSH. https://code.google.com/p/pdsh/
5) 田浦 健次朗, GXP : 分散環境を心地よく使う並列シェル, Vol. 27 (2010) No. 4 P 4_144-4_171 コンピュータ ソフトウェア
6) Nan Dun, Kenjiro Taura and Akinori Yonezawa, “GXP GMount: Building Ad-hoc Distributed File Systems by GXP and SSHFS-MUX”, Vol. 49 No. 4 IPSJ Journal Apr. 2008
7) 9front　http://9front.org/
8) Douglas Thain, Todd Tannenbaum, and Miron Livny, "Distributed Computing in Practice: The Condor Experience" Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, February-April, 2005