Vol.2017-OS-139 No.6
2017/3/1

# Plan9 の分散透明性を用いたファイルキャッシュと分散シェルによるコマンドベース MapReduce

中原健志[†1]　並木美太郎[†1]

**概要**：近年，ビッグデータの普及に伴い，様々な場所で分散処理が利用されている．分散処理のモデルとして MapReduce が有名である．しかし，MapReduce 向けフレームワークの多くでは，プログラムをフレームワークが提供する API を用いて改変する必要がある．本研究ではシェルのパイプ機能にデータ分割，Reduce 処理の機能を追加することで既存のコマンドによる MapReduce を実現し，短い開発期間で分散処理の効果を得ることを目指した．リモートマシン上でコマンドを実行する場合，ファイルやディレクトリの構成がローカルマシンと異なる課題がある．この課題を分散向け OS である Plan9 が持つ分散透明性の機能を用いて解決する．

**キーワード**：MapReduce，Plan9，Distributed Shell，分散シェル，コマンド

## 1. 研究背景

近年，コンピュータやネットワークの普及や性能向上に伴い，コンピュータが取り扱うデータ量が爆発的に増大している．2014 年 4 月に発表されたレポート[1]では，2013 年に世界で生成されたデータ量は 4.4 ゼタバイトであったと報告している．同報告書では今後も世界中で生成されるデータ量は増大の一歩を辿り，2020 年には 44 ゼタバイトに達すると予測している．近年，このような膨大なデータを高速に分析・処理する技術に注目が集まっている．大量のデータの処理では処理対象のデータサイズが大きいため，複数のマシンへデータや処理を分散して実行する分散処理が一般的に用いられている．分散処理では複数のコンピュータが必要となるが，クラウドサービスが普及したことで，分散処理を行う時のみに必要なコンピュータ資源をクラウドから購入することが出来るようになった．これにより分散処理に必要な環境を容易に準備できるようになった．一方で，生成される大量なデータの中には構造化されておらず，活用されていないデータも多い．近年，これまで収集された大量の構造化されていないデータを保存・分析し，データ間の相関や特徴などの有益な情報を見つけ，ビジネスや業務効率化などに活用する動きが広まっている．この動きは「ビッグデータ」と総称され，関心が高まっている．ビッグデータの処理では，大量のデータに対して様々な観点から分析などの処理を行う．この際，処理のパラメータなどを変更して，何度も試行を繰り返すことが多い．このため，高速かつ短期間で分散処理を実現する技術が求められている．これらビッグデータを処理する技術として「MapReduce[2]」と呼ばれるプログラミングモデルが広く利用されている．

## 2. 既存の MapReduce フレームワークの課題

MapReduce のオープンソースの実装として「Apache Hadoop[3]」と「Apache Spark[4]」が広く利用されている．また既存のコマンドを MapReduce 処理に応用できるフレームワークとして「Hadoop Streaming[3]」，「GXP[5]」について述べる．

### 2.1 Apache Hadoop

Hadoop では MapReduce の処理に分散ファイルシステムである HDFS を用いて，処理データの分散や処理結果の共有を行っている．HDFS は複数のノードから構成される特殊なファイルシステムであり，巨大なデータを取り扱うことに特化している．Hadoop では HDFS がデータを各マシンへ分散することで処理を並列に行うことを実現している．ただし HDFS は特殊なファイルシステムであり，OS へマウントしてプログラムから直接読み書きすることは出来ない．また，Hadoop では Map 処理と Reduce 処理に関する API を提供しており，ユーザはプログラムを Hadoop が提供する API を用いて書き換える必要がある．

### 2.2 Apache Spark

Apache Spark では処理の中間結果をメモリ上に保存し，次の処理へ直接渡すことで Hadoop に比べ中間データの受け渡しに要する時間を減らしている．Spark ではシェルが提供されており，ユーザはシェルから対話的に処理を実行することが可能である．ただし，Spark のシェルは通常の UNIX などシェルとは異なり，プログラミング言語の一種である scala や python のインタプリターに Spark の機能を追加したものである．このため，通常のシェルコマンドを MapReduce へ利用することは難しい．

### 2.3 Hadoop Streaming

Hadoop の拡張である「Hadoop Streaming」はシェルコマンドを MapReduce 処理に利用可能なフレームワークである．Hadoop Streaming では，Map 処理と Reduce 処理を別々のシェルコマンドとして定義し，それぞれのコマンドが持つ標準入出力を介してデータを受け渡すことでシェルコマンドによる MapReduce を実現した．この方式では標準入出力を持つプログラムであれば MapReduce の処理が出来るため，ユーザが普段利用しているシェルコマンドを容易

†1 東京農工大学
Tokyo University of Agriculture and Technology

に応用可能である．このため，開発に要する手間や時間を減らすことが可能である．一方で Hadoop Streaming では HDFS 上に処理対象のデータを配置する必要がある．しかし，HDFS は通常のプログラムからは直接アクセスすることが出来ないため，シェルコマンドから HDFS 上のファイルへ直接アクセスする処理を実現することが難しい．また，Hadoop Streaming で MapReduce のジョブへのデータ受け渡しは HDFS を介して行われる．このため，パイプを用いて他のシェルコマンドと連携した処理を行うことが難しい．

### 2.4 GXP

GXP は複数のクラスタに ssh を用いて接続し，指定したコマンドを実行する分散シェルである．GXP では各クラスタ上で指定されたコマンドを立ち上げ，それぞれのコマンド標準入力へ分割されたデータを送り，それぞれの処理結果をまとめることで分散処理を実現している．GXP はシェルコマンドを処理に用いるため，プログラミング言語の縛りなどがなく，ユーザが普段利用しているプログラムを分散処理に応用することが出来る．GXP には MapReduce の機能があり，Map の処理と Reduce の処理をユーザがコマンドやスクリプトとして作成することで MapReduce を実現することが出来る．GXP では，プログラムの配布は利用者に任せられる．そのため，利用者はあらかじめ各クラスタ間で共有できるファイルシステムを用意し，処理に必要なプログラムをそれぞれのクラスタへ配布する必要する必要がある．また，処理に必要なファイルが配置されているディレクトリがローカルマシンと同一であることを保証できない．

以上から下記を本研究の課題とする．

(1) **プログラム改変の手間が大きい**

Hadoop や Spark では MapReduce の処理をプログラム中でフレームワークが提供する API を用いて記述する必要がある．このため，ユーザは分散処理を行う際にプログラムを書き換える必要があり，普段利用しているプログラムをそのまま利用することが難しい．そのため，短い開発期間で分散処理の効果を得ることが難しい課題がある．

(2) **ローカルマシンとリモートマシンでファイルやディレクトリの構成が異なる**

既存の MapReduce 処理系では，ファイルやディレクトリの構成がローカルマシンとリモートマシンでは異なる．Hadoop や Spark では HDFS を用いることでローカルマシン，リモートマシン間でデータを共有することを実現している．しかし HDFS は特殊なファイルシステムであり，OS へマウントして様々なプログラムがアクセスすることが出来ない．一方 GXP では，ローカルマシン上にあるファイルをリモートマシンが直接参照することが出来ない．そのため，利用者は処理に必要なファイルを予めリモートマシンへコピーする，もしくは共有のファイルサーバなどへコピーし，リモートマシンへマウントする必要がある．しかし，多くの OS では一般ユーザは外部のファイルサーバやディレクトリをリモートマシンのファイルツリーへ自由に配置することが出来ない．このため，ローカルマシンとリモートマシン間でファイルツリーの共通の場所へファイルやディレクトリを配置することが出来ず，ユーザはリモートマシンをローカルマシンのファイルツリーと同様に扱うことが出来ない課題がある．

(3) **一般のプログラムから HDFS への参照が困難**

Hadoop や Spark では特殊な分散ファイルシステムである HDFS を用いてデータを各マシンへ分散している．HDFS では各マシンのローカルなディスクに分散したデータを配置することで処理対象のデータを複数のマシンで並列かつ高速に処理することを実現している．しかし，HDFS は特殊なファイルシステムであるため，通常のプログラムから直接参照することが出来ない課題がある．

## 3. 目標

本研究では 2 章で挙げた課題に対して次の 3 項目を目標とする．

### 3.1 既存のコマンドを MapReduce へ利用

現在，広く利用されている MapReduce フレームワークの多くでは，処理対象のプログラムをフレームワークが提供している API を用いて MapReduce 用に書き換える必要がある．本研究では，既存のコマンドを直接応用できる MapReduce フレームワークを提案する．長年利用されている既存のコマンドを再利用することで短い開発期間で分散処理の効果を得ることを目指す．また，開発言語の縛りを少なくし，ユーザが得意とするプログラミング言語で処理を記述することで開発にかかるコストを減らす．

### 3.2 複数のコマンドによる MapReduce 処理の実現

MapReduce の処理の多くは入力データに対するフィルタリング処理であることが多い．UNIX 系 OS が備えるシェルコマンドの多くはテキストデータに対するフィルタリング処理を得意とするものが多い．これらのシェルコマンドは単体が持つ機能は少なくシンプルであるが，複数のコマンドの入出力を繋ぎ処理を行うことで複雑な処理を実現している．本研究では複数のコマンドをシェルのパイプを用いて繋ぎ、MapReduce の処理に適用可能にする．これにより，様々なコマンドが連携した MapReduce 処理を実現する．

### 3.3 リモートマシンからローカルマシンのファイルへの透過なアクセスの実現

既存の MapReduce フレームワークの多くでは各マシンのディレクトリ構成やインストールされているプログラムがマシンによって異なる．このため，ユーザは利用するマシンによってディレクトリやインストールされているプログラムが異なることを意識してフレームワークを利用しな

ければならない. 本研究では分散環境向け Operating System である「Plan9」の分散透明性の機能を用いてマシン間で透過的に利用できる共有のディレクトリを提供する．また，ローカルマシンとリモートマシン間でのディレクトリやファイルの差異をフレームワークが隠蔽することで，リモートマシンであってもローカルマシンと同様に利用できる位置透過性を実現する．

### 3.4 データの分散とファイルキャッシュによる高速化

既存の MapReduce フレームワークでは分散ファイルシステムである HDFS を用いてデータの分散を行っていた．しかし，HDFS は特殊なファイルシステムであるため，通常のプログラムからは参照できない課題があった．本研究ではデータの分散をパイプを通して行うことで，様々なプログラムを MapReduce の処理へ応用可能にする．一方で HDFS では各マシンのローカルなディスク上に分散したデータを配置することで処理の高速化を実現した．本研究では各マシンに HDFS の代替としてファイルキャッシュの機能を持たせることで処理の高速化を実現する．

## 4. 分散向け OS「Plan9[6]」

Plan9 はベル研究所が開発した分散環境向けの Operating System（OS）である．Plan9 は複数のマシンがネットワークで接続されている環境を前提として設計された OS であり，下記の分散透明性の機能を持つ．

**(1) アクセス透過性**

Plan9 では様々なコンピュータ資源をファイルへ仮想化して提供している．ファイルに仮想化することで，様々なアプリケーションからファイルの read/write で資源を利用することが可能である．Plan9 のアクセス透過性の例として，ファイルの内容を表示する「cat」コマンドで資源状態を取得できる．また，ファイルへ文字列を書き込む「echo」コマンドで指定した文字列を書き込むことで，資源に対して制御をかけることが出来る．Plan9 ではファイル資源間の読み書きを共通したプロトコルである 9P（Plan9 Filesystem Protocol）で利用することが出来る．

**(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．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，ローカルマシン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．Plan9 では上記の分散透明性の機能を用いることで，ネットワークを介して接続したマシンのディレクトリを自身の名前空間へマウントし，その中の特定のディレクトリを自身のマシンの特定のディレクトリへ重ねて配置することが可能である．また，Plan9 では分散透明性の機能を用いて名前付きパイプを他のマシン共有することが出来る．

本研究で開発した分散シェルの実装では，Plan9 が持つ上記の分散透明性の機能を用いる．

## 5. 本研究における分散システムの概要

### 5.1 設計方針

本研究では目標の章での述べた「プログラム改変が少ない MapReduce」を実現する方法として，UNIX 系 OS のシェルが持つパイプの機能に注目した．本研究ではパイプをリモートマシン上で動作するコマンドとのデータ受け渡しが出来るよう拡張する．加えて，パイプにデータの分割，及び Map と Reduce の機能を加えることでコマンドによる MapReduce を実現する．これらの機能を実現するために本研究で開発したシェルを分散シェルと呼ぶ．

### 5.2 分散シェルの基本方針

分散シェルでは行いたい処理を粗粒度な単位に分割し，分割された処理内容を入力と出力を一つずつ持つコマンドに置き換える．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．

### 5.3 パイプ記号によるコマンドの実行方法の指定

通常のシェルではコマンドはシェルを実行しているマシン上で実行される．分散シェルではシェルを実行しているマシン上でコマンドを実行するだけでなく，ネットワークを介して接続した他のマシン上で実行する機能が追加されている．コマンドの実行方法の指定は通常のシェルのパイプ記号を拡張した記号を用いて指定する．また，コマンドを複数のマシン上で分散して実行する場合，コマンドへ MapReduce の処理を適用するために入力データの分割, 及び合成の方法を指定する．分散シェルで用いるパイプの記号は通常の UNIX のパイプ記号"|" に加え，３つの記号（//|, //{, //}）がある．

### 5.4 負荷分散

コマンドをリモートマシン上で実行する際にどのマシンで実行するかは分散シェルが決定する．分散シェルはリモートマシンでコマンドを実行する前にリモートマシンの負荷状況を調べ，負荷が小さいマシンにコマンドの実行を優先的に割り当てることで特定のマシンへ負荷が集中することを防ぐ．

### 5.5 位置透過性の確保

通常，ネットワークを介して接続したマシンでは，インストールされているコマンドやファイル・ディレクトリの構成は異なる．そのため，ユーザは利用するリモートマシンにインストールされているプログラムやファイル・ディレクトリの構成がローカルマシンのものと異なることを意識して利用する必要があった．本研究ではリモートマシン

の名前空間をローカルマシンの名前空間と重ね合わせ，置き換える．名前空間を重ね合わせることでリモートマシンの名前空間上にローカルマシンのファイルやディレクトリが出現する．リモートマシン上で動作するプログラムがローカルマシン上にあるファイルへアクセスした場合，ローカルマシンのファイルやディレクトリが自動的に参照される．プログラムからはファイルやディレクトリがリモートマシン・ローカルマシンのどちらにあっても，同様にアクセスすることが出来る．これによりリモートマシンでもローカルマシンと同じファイルやディレクトリの構成を実現し位置透過性を確保する．

### 5.6 リモートマシンでのファイルキャッシュ

分散シェルではリモートマシンの名前空間がローカルマシンのものと置き換わることでリモートマシンからローカルマシンのファイルへアクセスすることが出来，リモートマシン上での処理に利用することが可能である．しかし，リモートマシンでの処理で頻繁にローカルマシン上のファイルへアクセスする必要がある場合，ネットワークを介してアクセスするためリモートマシン上に存在するファイルに比べ，アクセスに時間を要する．この課題を解決するためにユーザが指定したファイルをリモートマシン上へファイルキャッシュすることでファイルの読出しにかかる時間を減らす．

### 5.7 システム構成

分散シェルではユーザが直接利用しているマシンをローカルマシンと呼び，ネットワークを経由して接続しているマシンをリモートマシンと呼ぶ．図１に分散シェルのシステム構成を示す．リモートマシンは外部からの接続を受け付け，認証後にプログラムなどを実行することが可能である．ユーザはローカルマシンからリモートマシンの IP アドレスを指定して接続する．接続後にローカルマシンからリモートマシンへ指定したコマンドを実行することが出来る．ユーザがローカルマシンの認証管理システムにリモートマシンの認証情報を予め登録しておくことで，ローカルマシン・リモートマシン間の認証は自動的に行われる．分散シェルではリモートマシンへ接続後，分散シェルのデーモンプログラムである実行デーモンをリモートマシン上で動作させる．実行デーモンはユーザが分散シェル上でリモートマシンへコマンドを実行する処理を記述した場合，分散シェルから実行するコマンドの情報を受け取り，指定されたコマンドを実行する．上記の方針に基づいて開発した分散シェルの設計について，次の章で詳細を述べる．

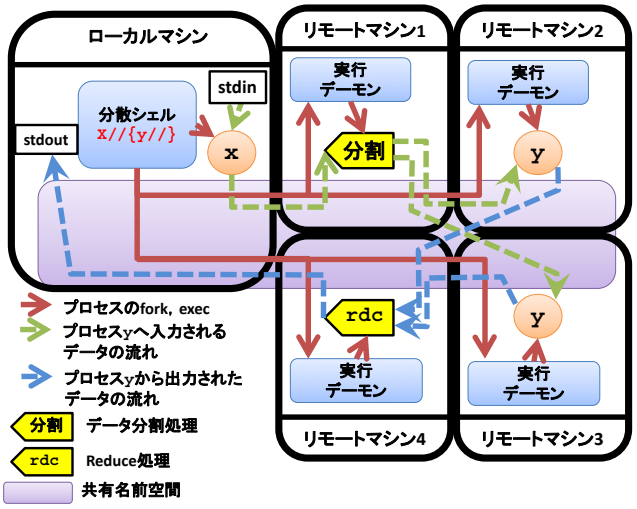


図 1 本研究で想定しているシステム構成図

## 6. 分散シェルの機能設計

本章では，分散シェルが提供する具体的な機能と分散シェルでの処理の流れについて説明する．

### 6.1 分散シェルが提供する機能

分散シェルはシェルコマンドをローカルマシン，リモートマシン上で動作させるために下記の機能を提供する

**(1) コマンドのリモート実行**

リモート記号で指定されたコマンドを１台のリモートマシン上で実行する．利用するリモートマシンは分散シェルが自動的に選択する．実行されるコマンドの標準入出力は分散シェルが自動的に前後のコマンド，またはファイルのものと接続する．

**(2) コマンドの MapReduce 実行**

MapReduce 記号で囲まれたコマンドは複数のリモートマシン上で分散して実行される．分散シェルは入力データを分割し，各リモートマシン上で実行されるコマンドの標準入力へ渡す．また，各リモートマシンで実行されたコマンドの処理結果は分散シェルが一つに纏めて出力する．

**(3) リモートマシンの名前空間の重ね合わせ**

コマンドをリモートマシン上で実行する場合，分散シェルはリモートマシンの名前空間をローカルマシンの名前空間と置き換える．リモートマシン名前空間がローカルマシンの名前空間へ置き変わることでユーザがリモートマシン上でコマンドを実行する場合，ローカルマシンのファイルが表示されリモートマシン上で利用可能になる．

**(4) リモートマシンへのファイルキャッシュの提供**

リモートマシン上で頻繁にアクセスするローカルマシンのファイルをユーザが分散シェル上で指定することでリモートマシン上にファイルが複製され，元ファイルと同じファイルパスへ配置される．ファイルの参照先がリモートマシン上へ複製したファイルへ変更されることでファイル読出しに要する時間を減らし処理の性能向上を図る．

**(5) 負荷分散**

コマンドをリモートマシンで実行する場合，分散シェルは利用可能なリモートマシンへ接続し，それぞれのマシンの負荷情報を収集する．収集された負荷情報を基にコマンドを実行するマシンの割り当てを行うことで特定のリモートマシンに負荷が集中することを防ぐ．マシンの負荷情報として CPU のアイドル時間を利用する．コマンド種別ごとのコマンド割り当てポリシーを示す．

(1) **リモート実行**

利用可能なリモートマシンから１台を選択する．選択方法として MapReduce 実行のコマンドがない場合，CPU のアイドル時間が長いマシンへ優先的に割り当てられる．もし，MapReduce 実行のコマンドがあった場合，Map コマンドが実行されるマシンの中から CPU のアイドル時間が長いマシンが優先的に割り当てられる．

(2) **MapReduce 実行**

**I. データ分割コマンド**

データ分割コマンドの前に実行されるコマンドがあった場合，データ分割コマンドは前に実行されたコマンドと同じマシン上で実行される．これは前のコマンドのデータ出力を同じマシン上で受け取ることでデータ転送のオーバヘッドを減らすためである．もし，コマンドの前に他に実行されるコマンドが無かった場合，分散シェルへ入力されるデータはローカルマシンから入力されることが多いため，ローカルマシン上でデータ分割コマンドを実行する．

**II. Map コマンド**

Map コマンドは Reduce 実行で選択されたマシン以外とローカルマシンを除いたすべてのリモートマシン上で実行される．

**III. Reduce コマンド**

Reduce コマンドは CPU アイドル時間が最も長いリモートマシン上で実行される．もしくはユーザがマシンの IP アドレスを指定して，実行することも可能である．

### 6.2 処理の記述方法

#### 6.2.1 分散シェルの記法と概要

分散シェルでは通常の UNIX シェルの記号に追加した３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．追加したそれぞれの記号の役割について説明する．

(1) **リモート実行**

```
//| cmd
```

リモート記号では cmd で指定されたコマンドを１台のリモートマシン上で実行する．実行するマシンは分散シェルがリモートマシンの処理負荷の状況を調べた上で決定する．リモート記号で指定されたコマンドの前後に他のコマンドやファイルが指定されていた場合，分散シェルはコマンドへの標準入出力を自動的に他のコマンドやファイルの標準入出力へつなげる．

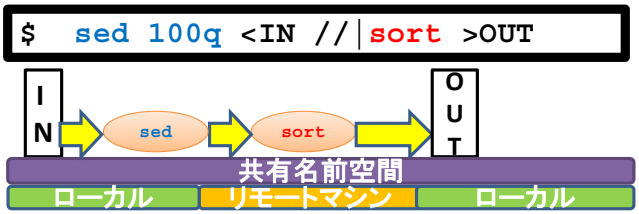


図 2 リモート記号での処理概念図

図２の例では，コマンド「sed 100q」はローカルマシン上で実行される．次のリモート記号で指定されたコマンド「sort」は，分散シェルが選定した１台のリモートマシン上で起動される．この際，それぞれのコマンドの入出力は分散シェルにより，自動的に繋がれる．

(2) **MapReduce 実行**

```
//{[split 方法] cmd [reduce 方法]//}
```

MapReduce 記号では cmd で指定されたコマンドを利用可能な複数のリモートマシン上で分散実行する．コマンドへの入力データは split 方法で指定された方法で分割され，各リモートマシン上で実行されるコマンドの標準入力へ渡される．複数のリモートマシン上で実行されたコマンドの標準出力は指定された reduce 方法で一つに纏められ，出力される．分割方法の指定がない場合，分散シェルは入力データを行単位で分割して，各リモートマシンへ渡す．これは多くのシェルコマンドが改行を区切りとした行単位のテキスト処理に向いた設計となっているためである．また，reduce 方法の指定がない場合，分散シェルは各リモートマシンから出力された順番でデータをまとめていく．処理の内容によっては入力データの分割方法や出力データの reduce 方法をユーザが追加したい場合が考えられる．この場合ユーザ独自の split 方法，reduce 方法を追加し，指定することが可能である．

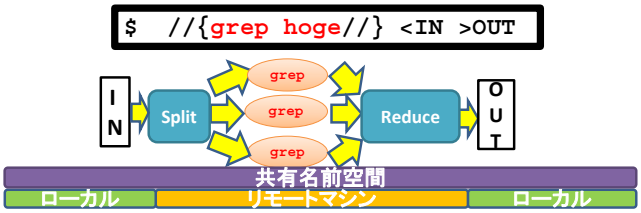


図 3 MapReduce 記号での処理概念図

図３の例では，MapReduce 記号で囲まれたコマンド「grep」は複数のリモートマシン上で起動される．入力データは Split 処理により改行単位で分割され，それぞれのリモートマシン上の「grep」へ渡される．処理された結果は分散シェルにより Reduce 処理が行われ，一つのデータに纏められて出力される．

## 7. 分散シェルの内部設計

分散シェルはフロント管理部と実行デーモンから構成される．フロント管理部ではユーザからコマンドの入力を受け付け，解釈し，マシンの割り当てや通信路の設定など

を行う．コマンドの解釈や通信路の設定が完了するとフロント管理部はコマンドを実行するマシンに対して，実行デーモンを立ち上げ,通信路の確立や処理の実行を指示する．実行デーモンはフロント管理部から指示を受けて通信路や共有名前空間の生成をする．そして，通信路などが確立した後に指定されたコマンドの標準入出力を変更し実行する．

### 7.1 フロント管理部

フロント管理部は分散シェルを立ち上げた際に最初に起動されるプログラムである．フロント管理部はユーザからのコマンドの入力を受け付け，実行種類によってコマンドを分別する「パーサー部」, 及びパーサー部が分別したコマンドとそれぞれのコマンドの標準入出力と通信路を管理する「コマンド管理部」, 及びコマンドを実行するマシンを決定し管理する「実行マシン管理部」から構成される．

### 7.2 実行デーモン

実行デーモンはフロント管理部が起動し，マシンに割り当てられたマシン番号，実行するコマンド，及び生成する名前付きパイプの情報が渡される．実行デーモンはこれらの情報を基に必要な名前付きパイプを生成し，ネットワーク上へ公開する．また，他のマシンからデータを受け取る必要がある場合，それぞれのマシンの名前付きパイプを自身のマシンの名前空間へマウントする．他のマシンの名前付きパイプを自身の名前空間へマウントすることで他のマシン上で動作するコマンドと名前付きパイプを通してデータのやり取りを行うことが可能になる．また，自身のマシンの指定されたディレクトリ以下の名前空間をローカルマシンのものに置き換える．これにより，リモートマシン上で動作するコマンドが名前空間を置き換えたディレクトリへアクセスした場合，ローカルマシンのファイルへ容易にアクセスすることが可能である．実行デーモンは名前付きパイプ管理部，名前空間管理部，キャッシュ管理部，及びコマンド実行部から構成される．キャッシュ管理部について，述べる．

### 7.3 キャッシュ管理部

キャッシュ管理部ではユーザが指定したファイルをリモートマシン上でファイルキャッシュとして提供する．実行デーモンでは名前空間管理部がリモートマシンの名前空間をローカルマシンの名前空間へ置き換えることで，ユーザがリモートマシン・ローカルマシン間で透過にファイルを扱うことを可能にしている．しかし，実行されるコマンドによっては処理の際に辞書ファイルといった他に参照するファイルが必要なものがある．処理中に頻繁に外部のファイルを参照する場合，ローカルマシンとリモートマシン間はネットワークを介してファイルの参照を行っているためオーバヘッドが大きく，参照に時間がかかり処理時間が長くなってしまう課題があった．そこで，キャッシュ管理部ではローカルマシンから指定されたファイルをファイルキャッシュとしてリモートマシン上に複製する．ファイルの複製が完了後，位置透過性の機能を用いて指定された元ファイルをファイルキャッシュのものへ置き換える．これにより，ローカルマシンの名前空間を参照しつつ，一部のファイルについてはコマンドを実行するマシン上に複製されたファイルへ参照先を変更することでファイルアクセスにかかる時間を減らし，コマンド全体の処理時間を短くする．一方で実行するコマンドからは参照しているファイルがローカルマシン・リモートマシンのどちらにあるかを意識せずに利用できることを目指す．

## 8. 実装

本研究で設計した分散シェルを Plan9 の派生 OS である 9front を用いて実装した．実装について述べる．

### 8.1 マシン間の通信路の生成

Plan9 では一般的な OS と異なり，名前付きパイプをネットワーク介して利用することが可能である．本研究では複数のマシン上で実行されるコマンド間の通信を Plan9 の名前付きパイプを用いることで実現した．また，名前付きパイプをネットワークへ公開する方法として，Plan9 の exportfs コマンドを用いて実現した．

### 8.2 名前空間の重ね合わせ

Plan9 では名前空間を編成する命令として bind を提供している．bind では指定した２つの名前空間を重ね合わせる，または置き換えることが可能である．bind を用いて名前空間を重ね合わせる際，もしファイルやディレクトリの名前が衝突した場合, bind ではどちらを優先するか設定することが可能である．分散シェルでは bind 命令を用いてリモートマシンとローカルマシンの名前空間の重ね合わせ, 複数のマシンからマウントした名前付きパイプの共有などを実現している．

### 8.3 キャッシュの実現

キャッシュの実装は，Plan9 の名前空間の重ね合わせの機能を用いて実現した．図４に分散シェルでのキャッシュの処理の概念図を示す．ユーザが分散シェル上でキャッシュを設定した場合，指定されたローカルマシン上のファイルをリモートマシン上で動作する実行デーモンがコマンドの実行前にリモートマシンのキャッシュ保持のディレクトリ以下へ複製する．複製完了後にリモートマシン上の名前空間でローカルマシンと名前空間が重ね合わせられているディレクトリ下のキャッシュの元ファイルをリモートマシンへ複製したファイルの名前空間と置き換える．これによりリモートマシンからキャッシュに指定されたファイルを読み込む場合，ローカルマシン上の元ファイルと同じファイルパスでリモートマシンの複製されたファイルへアクセスが行われる．キャッシュしたファイルを参照するコマンドは Plan9 の分散透明性の機能からファイルの参照先がローカルマシン，リモートマシンのどちらであるかを意識せず利用することが可能である．分散シェルがファイルキャ

ッシュを提供することでファイルの読出し先がリモートマシンのローカルディスクになり，ファイルの読出しにかかる時間が減ることでコマンドの処理性能を上げることが出来る．

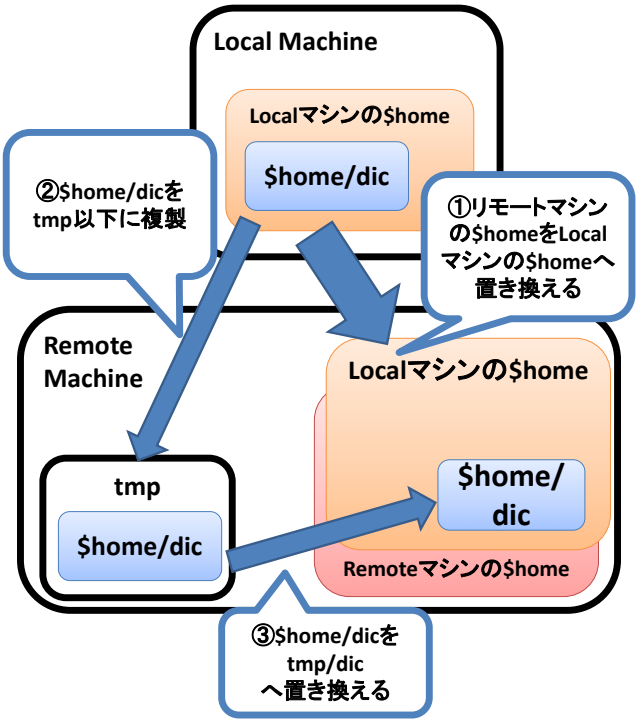


図 4：Plan9 の透過性によるファイルキャッシュの処理概要

## 9. 評価

分散シェルを用いてコマンドによる MapReduce を実行した場合の性能評価として下記の評価を行った．

(1) **WordCount の MapReduce 実行時の性能評価**

(2) **ファイルキャッシュによる性能評価**

評価環境として，ローカルマシン１台，Reduce マシン１台，Map 処理用のマシンを４台用いた（図５）．評価に用いたマシンのスペックを表に示す．

| | |
|---|---|
| CPU | Intel Corei7-2600@3.6Ghz |
| メモリ | 8GB |
| ストレージ | HDD（500GB） |
| OS | 9front(2016 /10 /27 リリース版) |

表 1：評価に用いたマシンのスペック

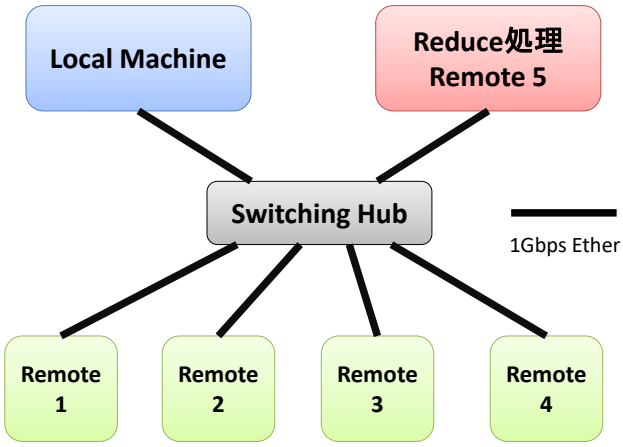


図 5：評価環境の構成図

### 9.1 WordCount による MapReduce 実行時の性能評価

テキストの単語出現数を数える処理例を用いて，処理に要する時間を計測した．また，評価に用いたコマンドの中で単語を分解する word コマンドは自作のコマンドであり，リモートマシンへはインストールされていない．word コマンドはホームディレクトリ上に配置してある．分散シェルがリモートマシンに接続する際にリモートマシンのホームディレクトリをローカルマシンのものへ置き換える．これにより，分散シェルのカレントディレクトリがホームディレクトリである場合，word コマンドのファイルがホームディレクトリ上に表示され，実行することが可能である．処理に用いたコマンドを表２に示す．また，評価結果を図６に示す．

**＜評価に用いたコマンド＞**

```
cat $home/input.txt //{word | sort [merge]//}
uniq -c >/dev/null
```

表 2：WordCount の評価に用いたコマンド

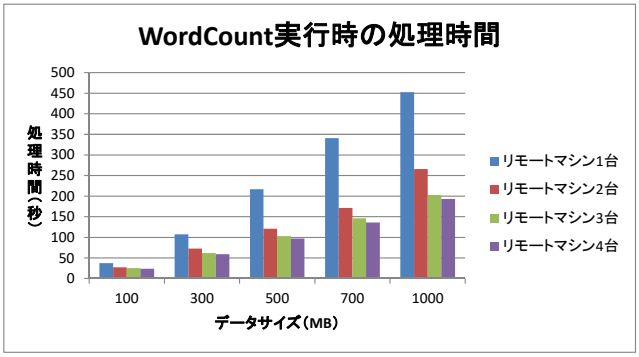


図 6：WordCount 処理に要した時間と Map マシンの台数関係

リモートマシンでの処理にローカルマシン上にしか存在しないコマンドを利用した場合でも，分散シェルの名前空間を共有する機能で問題なく利用できることが確認出来た．これにより，リモートマシンでもローカルマシンと同

様に扱える環境を提供できていることが確認できた．性能の向上では Map 処理に利用するリモートマシンの台数が１台の場合の処理時間を基準としたとき，２台で最大 1.99 倍, ３台で 2.34 倍, ４台で 2.51 倍の性能向上比となった．このことからコマンドによる MapReduce を用いて性能が上がっていることを確認出来た．

### 9.2 ファイルキャッシュによる性能評価

辞書データを必要とする検索コマンドである「look コマンド」を用いて，ファイルキャッシュの性能向上について評価を行った．評価に用いたコマンド列を表３に示す．

**＜評価に用いたコマンド列＞**

```
cat input.txt //{word｜look -ixd $home/dic //}
>/dev/null
```

表 3：ファイルキャッシュの評価に用いたコマンド列

cat コマンドでは１ MB のテキストファイルである「input.txt」 の内容を出力する．出力された内容は MapReduce 記号で囲まれたコマンドへ行単位に分割されて各リモートマシン上で実行される word コマンドへ渡される．word コマンドでは入力されたテキストを単語に分解し，look コマンドへ渡す．look コマンドは指定された辞書ファイルである dic の中に登録されている単語と入力された単語を比較し，合致する場合のみ出力する．本評価では word コマンド，及び辞書ファイルである dic はローカルマシンのホームディレクトリ上に配置されており，リモートマシン上には存在しない．dic のファイルサイズは 242KB である. 分散シェルでは名前空間の共有の機能からリモートマシン上でもローカルマシンのファイルやコマンドを利用することが可能である．そこで本評価では処理に必要なファイル「dic」 をキャッシュを用いずに利用した場合とキャッシュを用いて利用した場合での全体の処理時間を計測した．処理に用いた Reduce マシンは４台である．評価結果を表４に示す．

| キャッシュの有無 | 処理時間（秒） |
|---|---|
| キャッシュ有効 | 15 |
| キャッシュ無効 | 184 |

表 4：キャッシュを用いた場合の処理時間

キャッシュを用いた場合が用いなかった場合と比べ 12 倍ほど処理時間が短い結果となった．これは look コマンドでは単語がコマンドへ入力されるたびに辞書ファイルの中を２分探索しており，ファイル I/O が持つバッファリングの効果が出にくいため，ネットワークを介した辞書参照のオーバヘッドが大きくなり処理時間が長くなったと考えられる．分散シェルで提供しているキャッシュは透過に配置されているため，ユーザはキャッシュをローカルマシン上に配置されているファイルと同様に取り扱うことが可能である. 評価結果から MapReduce 処理でコマンドが他に参照するファイルがある場合にキャッシュが性能向上に有効であることが確認できた．

## 10. まとめ

本研究では，シェルのパイプにデータの分割，及びデータの集約の機能を付加することで既存のコマンドによる MapReduce を実現するための分散シェルを提案, 開発した．また，分散透明性の機能を持つ OS「Plan9」の透過性の機能を用いることでファイルの実際の位置をプログラムから隠蔽し，ファイルがローカルマシン上であっても，リモートマシン上であっても透過に利用できる環境を実現した．加えて，Plan9 の分散透明性を用いたファイルキャッシュがローカルマシン上のファイルを参照する必要があるコマンドを実行する場合に有効であることが確認出来た．

今後の課題としてデータ分割，及び Reduce 処理の高速化が挙げられる．また，現在の設計では Reduce の処理は１台のマシンが担当する設計となっている．分散シェルで取り扱うリモートマシンの台数が増えた場合, Reduce を担当しているマシンへの処理負荷が集中してしまう．このため，データの分割，及び Reduce の処理を複数のマシンへ分散できる設計が必要である．その他にコマンドのリモートマシンへの割り当てについて，ネットワークの性能やコマンド間の依存関係を考慮したコマンドの実行マシン割り当てが必要である

## 参考文献

[1] IDC ’s Digital Universe, 「The Digital Universe of Opportunities:Rich Data and the Increasing Value of the Internet of Things」Sponsored by EMC（2014/4）

[2] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: simplied data processing on large clusters. In Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume6 (OSDI'04), Vol. 6. USENIX Association, Berkeley, CA, USA, 10-10.

[3] Apache Hadoop http://hadoop.apache.org/

[4] Apache Spark https://spark.apache.org/

[5] 田浦健次朗. (2010). GXP: 分散環境を心地よく使う並列シェルコンピュータソフトウェア, 27(4), 4 144-4 171.

[6] R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, and P.
Winterbottom. Plan 9 from Bell Labs. Computing Systems,