分散計算機環境における透過的なコマンド実行を可能とするコマンドインタプリタ

本田 治 † 多田 知正 † 樋口 昌宏 ‡ 井上 克郎 †

† 大阪大学
大学院基礎工学研究科 情報数理系専攻

‡ 近畿大学
理工学部 電気工学科

本稿では，透過的に遠隔計算機上のコマンドを実行できるコマンドインタプリタについて述べる．UNIXによる分散計算機環境では，NFS等の分散ファイルシステムを用いることによって，ファイルに対し透過的アクセスが可能である．しかし，コマンドに関しては，実行できる計算機が決まっているため，たとえ実行ファイルを共有しても，コマンドを透過的に実行することはできない．本研究で作成したシステムでは，移動エージェントの手法に基づき，計算機間を移動することで，遠隔計算機上のコマンドの実行を可能としている．また，複数の計算機上に同じコマンドがあるとき，適切な計算機を選択する．

A Command Interpreter which Enables Transparent Command Execution in Distributed Environment

Osamu Honda † Harumasa Tada † Masahiro Higuchi ‡ Katsuro Inoue †

†Graduate School of Engineering Science
Osaka University

‡School of Science and Engineering
Kinki University

In this paper, we describe about a command-interpreter which can transparently execute commands on a remote host. In the case of files, NFS, a popular distributed file system on UNIX, provides transparent accesses. As for commands, however, it is not the case. Though executable files at the remote host can be accessed through NFS, it generally cannot be executed at the local host. It means that NFS cannot achieve transparent accesses to remote commands. Our system uses mobile agent technology to move the command to the remote host and execute it on the host. Moreover, our system can select an appropriate host when the same command is available on several hosts.

1. はじめに

近年複数の計算機をネットワークでつないだ分散計算機環境が一般的となっている．分散計算機環境においては，ユーザは遠隔計算機上の資源をその物理的な位置を意識することなく利用できることが望ましい．これを遠隔資源への透過的なアクセスと呼ぶ．

ファイルについてはNFSなどの分散ファイルシステムによって透過的なアクセスがある程度実現されている．しかし遠隔計算機上のコマンドを利用するには，rlogin等で目的の計算機にログインするか，目的の計算機上でコマンドを実行し，プロセス間通信を用いて結果を返すプログラムを記述するという方法しか提供されていない．

コマンドもまた資源と考えると，透過的なアクセスが実現されているとは言えない．そこで，本研究では，ユーザが分散している資源をその位置を意識することなく利用するためには，位置透過性とアクセス透過性が必要であると考え，既存のOS(UNIX)上に，位置透過かつアクセス透過な形式で記述されたコマンドを解釈，実行するコマンドインタプリタの実装を行った．実装したシステムでは，コマンドのインポートおよびエクスポートという概念を導入した．また，コマンドが複数の計算機で実行可能である場合，コマンドの実行中に参照されるファイルの位置に着目して，適切な計算機上のコマンドを選択し，実行する機構を実装した．また，遠隔計算機上のコマンドに対してアクセスするために，移動エージェントの仕組みを採用している．これにより，実装したシステムは，ネットワークへの負荷が少ないといった移動エージェントの利点[1, 2, 3, 8, 9]をもつ．実装したシステムの移動のオーバヘッドおよびコマンド選択機能の有効性を評価するため，実際にスクリプトを記述し実験を行った．

以下では，2節で分散計算機環境について述べ，そ

の問題点を指摘する．3節では，実装したシステムの概要について述べ，4節では，システムの実装について述べる．5節では，実装したシステムを用いて行った実験について述べる．最後に6節でまとめを行う．

2. 分散計算機環境と問題点

2.1 分散計算機環境

現在，複数の計算機がネットワークを介して接続されている分散計算機環境が一般的である．分散計算機環境では，各計算機が何らかの資源を保持している．本論文では，資源を「何らかのデータおよびサービスを提供する物理的もしくは論理的実体」と定義する．例えば，スキャナやプリンタ等のハードウェアデバイスや，テキスト，画像データ，実行形式のプログラムなどを格納するファイルは資源である．ユーザは，いくつかの計算機に存在する様々な資源を利用して，目的の仕事を行う．

分散計算機環境では，遠隔計算機上に存在する資源 (遠隔資源と呼ぶ) に直接アクセスすることはできない．したがって，遠隔資源にアクセスするための何らかの機構が必要となる．

2.2 透過性

分散計算機環境は，ユーザには全体で一つのシステムに見えることが理想である．この性質を分散計算機環境の透過性 (transparency)[4] という．透過性は，位置透過性 (location transparency)，アクセス透過性 (access transparency)，複製透過性 (replication transparency) 等いくつかに分類される．ユーザが遠隔計算機上の資源をその物理的な位置を意識することなく利用できるためには，位置透過性とアクセス透過性が必要である．位置透過性は，ユーザが資源を参照する際に資源の位置を意識する必要が無いことを意味しており，これはさらに名前透過性とユーザモビリティに分類される．名前透過性とは，資源の名前が資源の位置に関する情報を含まないことであり，ユーザモビリティとは，ユーザがどの計算機からでも資源を同じ名前で参照できることをいう．またアクセス透過性とは，どの位置にある資源も同じ方法でアクセスできることをいう．本研究では，位置透過でありかつアクセス透過な分散計算機環境の実現を目的とする．

2.3 UNIX による分散計算機環境

本研究での対象は，UNIX ワークステーションで構成される分散計算機環境とする．UNIX 環境では，多くの資源をファイルの形で利用することができる．また，遠隔計算機上のファイルにアクセスするための手法が，いくつか提供されている．NFS は，現在 UNIX で広く用いられているファイル共有機構である．NFS では，自分の計算機のディレクトリ木に，遠隔計算機のディレクトリ木をマウントすることにより，遠隔計算機上のファイル (遠隔ファイルと呼ぶ) を，あたかも自分の計算機のファイルシステム上のファイルのようにアクセスすることができる．NFS は，名前透過性とアクセス透過性を満たす．さらに，各計算機のディレクトリ木の構造を同じにすることによって，ユーザモビリティも満たすことが可能である．また，Coda[7] は現在普及しつつある分散ファイルシステムである．Coda は，NFS と同様に名前透過性とアクセス透過性を満たす．また，全てのクライアントでファイル名の名前空間を共有することにより，ユーザモビリティも満たしている．このように UNIX では，ファイルに関してはすでに透過的なアクセスが提供されていると言える．

ここで，UNIX 上で実行されるコマンドについて考える．ユーザはコマンドを実行することで，何らかのデータあるいはサービスを得る．すなわち，コマンドは一種の資源であり，コマンドの実行は，コマンドという資源へのアクセスと言い換えることができる．以降では曖昧さをさけるため，資源としてのコマンドをコマンド資源と呼び，コマンド資源にアクセスする際にユーザが記述する文字列をコマンド名と呼ぶ．

分散計算機環境において，コマンド資源は，実行されるプログラムと，プログラムを実行する計算機によって特定される．よって，実行可能ファイルを NFS を用いて複数の計算機で共有しても，それは一つのファイルという資源であるが，各計算機でそのファイルを実行する場合には，それぞれは別のコマンド資源とみなす．

従来の UNIX においては，ローカルなコマンド資源については，プログラムを格納しているファイルのファイル名のみによって資源を特定できる．これは全てのコマンドはユーザがログインしている計算機上で実行されることが前提となっているためである．

一方 UNIX では遠隔計算機上に存在するコマンド資源 (遠隔コマンド資源と呼ぶ) を利用することはもともと想定されていない．このために，遠隔コマンド資源を利用するには煩雑な手続きが必要となる．一般的には，UNIX で遠隔計算機 *RH* 上のコマンド資源にアクセスするためには，サーバを *RH* 上に配置し，プロセス間通信を用いて *RH* 上のサーバと通信する方法を用いる．サーバは資源アクセスの要求を受け，遠隔コマンド資源へのアクセスを行い結果を返す．この方法は，`rlogin` や `telnet` によるリモートログイン，`rsh` 等の既存のサーバプログラムの利用，ユーザによるサーバプログラムの作成の主に3種類が存在する．しかしながら，これらの方法は位置透過性やアクセス透過性を満たさない．

2.4 UNIX による分散計算機環境の問題点

2.3節で述べたように，従来の UNIX による分散計算機環境では，ユーザは必要なコマンド資源がどの計算機上にあるのかを把握していなければならず，また，遠隔コマンド資源にアクセスする場合は，ローカルなコマンド資源にアクセスする場合と違う手順をとる必要がある．これはユーザにとって大きな負担となる．また，あるコマンド名が複数の計算機上のコマンド資源に対応する場合，計算機の選択はユー

ザに委ねられることになる．分散計算機環境の構成によっては，コマンドを実行する計算機によって，実行速度や，ネットワークへの負荷が大きく異なる場合がある．もしユーザが適切でない計算機を選ぶと，全体の性能に影響を与える可能性がある．よって，計算機の選択は，ユーザではなく管理者によってなされることが望ましい．

3. システムの概要

2.4節で述べた問題を解決するために，UNIXによる分散環境で，ユーザが透過的に記述したコマンド名又はスクリプトを解釈し実行するためのシステムを設計し，実装した．

実装したシステムでは，複数の遠隔計算機のコマンド資源を利用できなければならない．そのためには，それぞれの計算機のコマンドの名前空間を結合する必要がある．NFSではディレクトリ木のマウントという仕組みで，ファイルの名前空間を結合している．一方，コマンド名の集合はファイル名のように階層構造を持たない．また，ファイルのパス名がファイルを一意に指定するのに対し，コマンド名は複数の計算機のコマンド資源に対応する．このため，コマンド資源を共有するには別の仕組みが必要となる．

3.1 コマンド資源の位置透過性

コマンド資源を共有する場合，一つの方法として，分散計算機環境全体のコマンド資源に関する情報を，すべての計算機で共有するという方法が考えられる．これは，Codaに代表される分散ファイルシステムで用いられる方法であり，これにより，常に位置透過性が保証される．しかし，コマンド資源の共有には，この方法は必ずしも適切ではない．分散ファイルシステムの場合，ファイル名はただ一つのファイルを表す．よって，位置透過性を保証することは，すべての計算機上で，一つのファイル名が同じファイルに対応することを意味する．一方コマンド資源の場合，一つのコマンド名は，一般に複数の計算機上のコマンド資源に対応し，それらはユーザからみて等価である．すなわち，コマンド資源の位置透過性を考える場合，各計算機上で，コマンド名に対応するコマンド資源は異なっていてもよい．むしろ，分散計算機環境において，すべての計算機が同じコマンド資源を参照することは，負荷の集中を招く場合がある．よって，それぞれの計算機上でコマンド名がどのコマンド資源に対応するかを管理者が指定できることが望ましい．

そこで本システムでは，NFSと同様に計算機ごとにコマンド資源の情報を管理する方式を採用する．本システムでは，コマンド資源のエクスポートとインポートという概念を導入する．各計算機はコマンド資源の集合をエクスポートし，それぞれの計算機は必要なコマンド資源をインポートすることで，その計算機がアクセスするコマンド資源の集合を構成する．

表 1　コマンドエクスポートテーブル

```
/bin/arch,df,mount,ps
/bin monblanc monch
/usr/local/bin/xdvi
/usr/bin 133.1.236.0/255.255.255.0
/usr/local/bin
/usr/bin/dvips monblanc monch
/usr/jdk1.3/bin *
```

3.2 コマンド資源のエクスポート

計算機 H がコマンド資源をエクスポートするとは，H が H 上で実行可能なファイルをコマンド資源として利用することを宣言することである．その際，そのコマンド資源の利用を許可する計算機も宣言する．コマンド資源をエクスポートするには，コマンドエクスポートテーブル/etc/cmdexportsに登録する．コマンドエクスポートテーブルの例を表1に示す．一つのエントリはエクスポートするコマンドに資源対応するファイル名とエクスポートする計算機名のリストからなる．ファイル名はディレクトリの後に":" で区切って書くことができる．ファイル名の代わりにディレクトリを指定した場合は，そのディレクトリの全ての実行可能ファイルを指定したことになる．計算機名のフォーマットはNFSのエクスポートファイル/etc/exportsに準ずる．計算機名のリストが空のときは，自分自身にのみエクスポートされる．コマンド資源は，エクスポートテーブルの上のエントリから順番に検索され，最初に見つかったコマンド資源がそのエントリで指定された計算機にエクスポートされる．すなわちエクスポートテーブルは従来のコマンド検索パスの役割を果たす．表1の場合，/bin以下のコマンド資源はmontblancとmonchにエクスポートされるが，archやdfのようなコマンド資源は自分自身にのみエクスポートされる．

3.3 コマンド資源のインポート

計算機 H がコマンド資源をインポートするとは，H 自身または他の計算機によってエクスポートされたコマンド資源を H で利用すること，すなわち H でのコマンド資源の検索の対象に含めることを宣言することである．エクスポートされたコマンド資源は，インポートすることで実際に利用できるようになる．コマンド資源をインポートするには，コマンドインポートテーブル/etc/cmdimportsにコマンドを登録する．コマンドインポートテーブルの例を表2に示す．一つのエントリはインポートする計算機名とインポートするコマンド名のリストからなる．表2の4行目は自分自身のエクスポートしているコマンド資源をインポートすることを表す．5行目は計算機monchのエクスポートしている全てのコマンド資源をインポートすることを表す．6行目のように，コマンドのリストの先頭が"−" の時は，jungfrauがエクスポートしているコマンド資源のうち，リス

表 2　コマンドインポートテーブル

```
everest wavcompress
eiger.ics.es.osaka-u.ac.jp convert,a2ps
>
local
monch
jungfrau -a2ps,convert
>
eiger
```

ト中の (a2ps と convert) を除いた全てのものがインポートされる．コマンド名の代わりにディレクトリを書くことはできない．コマンドインポートテーブルはまた，コマンド資源検索の際の優先度を指定する．テーブル中の連続する 2 つの行に対応するエントリは同じ優先度を持つ．“>” のみからなる行はその上のエントリの優先度がその下のエントリの優先度よりも高いことを表す．

3.4　コマンド資源検索

計算機 H におけるコマンド名 N に対するコマンド資源検索は以下の手順で行われる．システムは N を受け取ると，自身のコマンドインポートテーブルを参照し，エントリの存在するすべての計算機に対して，N に対応するコマンド資源を H にエクスポートしているかを問い合わせる．問い合わせを受け取った計算機では自身のコマンドエクスポートテーブルを元に，N に対応するコマンド資源を検索し，H に返答する．H ではすべての返答が得られるか，設定したタイムアウト時間が経過するまで，返答を待つ．得られたコマンド資源の集合は，以降のアクセスのためにキャッシュする．

3.5　コマンド資源の選択

システムがコマンド資源検索を行い，複数の計算機のコマンド資源が得られた場合，システムがそのうちの一つを選択してアクセスを行うが，コマンド資源によっては，どの計算機のコマンド資源を選択するかによって，実行時間やネットワーク負荷等が大きく異なる場合がある．

コマンド資源の選択方針の一つに，コマンド資源がアクセスするファイルに着目することが考えられる．NFS で共有されたファイルへのアクセスは通信を伴うため，自分自身が保持しているファイルへのアクセスに比べてコストがかかる．コマンドの引数となっているファイルが保存されている計算機のコマンド資源を優先的に選択することでコストを抑えることが可能であると考えられる．本システムの現在の実装では，この方針に基づいてコマンド資源の選択を行う．しかしコマンド資源の選択方針には，他にもさまざまなものが考えられる．今後，他の選択方針を実装し，コマンド資源ごとに最適な選択方針を設定できるようにする予定である．

3.6　コマンド資源へのアクセス方法

コマンドインタプリタの遠隔コマンド資源へのアクセス方法について述べる．

計算機 RH の遠隔コマンド資源を用いるには，なんらかの形で RH 上でプログラムを実行しなければならない．

コマンド資源へのアクセス方法として，もっとも単純なものはコマンド名と引数を遠隔計算機上のサーバプロセスに送り，実行結果を受け取る方法である．別の方法として，移動エージェントの仕組みを利用することが考えられる．移動エージェントとは，計算機間を移動しながら，目的の仕事を達成するプログラムである．移動エージェントは，プログラムの変数やスタックの値等の内部状態を保持することにより，計算機間を移動しても移動前の計算機で中断したところから，プログラムの実行を再開することができる．移動エージェントは近年盛んに研究が進められており [1, 2, 3, 8, 9]，幾つかのシステムは既に実装されている．そこで，ユーザによって与えられたコマンド名，あるいはスクリプトが，移動エージェントとして遠隔計算機に移動し，コマンド資源にアクセスする方法が考えられる．

この方法には，以下のような利点が存在する．すなわち，移動先の計算機でその後の処理を引続き行えるため，コマンドの実行結果をネットワークを介して転送する必要がない．これにより，実行結果がサイズの大きなデータの場合にネットワークへの負荷を減らすことができる．また，遠隔計算機 H 上のコマンド資源へのアクセスが連続する場合には，一度 H に移動するだけでよい．個々のコマンド名を H 上のサーバに送る方法と比べて，実行速度やネットワークへの負荷の点で有利である．

逆に，不利な場合も存在する．エージェントの移動は，一般的にサーバへのアクセス要求に比べると通信量が大きい．比較的扱うデータのサイズが小さく，全体の実行時間の短いようなコマンド資源に関しては，エージェント移動のためのコストがコマンド資源の実行全体に占める割合が大きくなる場合がある．しかしながら，スクリプトのような比較的小規模のプログラムを移動エージェントとして実行した場合，その内部状態を移動するための通信量はさほど大きいものでなく，マルチメディアデータなどと比べればはるかに小さい．

以上のことを考慮して，本システムでは，遠隔コマンド資源へのアクセスに，移動エージェントの仕組みを利用することにした．

4.　実装

ここでは，実装したシステムの詳細について述べる．

4.1　構成

本システムでは，スクリプトが移動エージェントとして移動する必要があるため，システムの基本的な構成は既存の移動エージェントシステムと同様である．すなわち，各計算機上に移動エージェントの

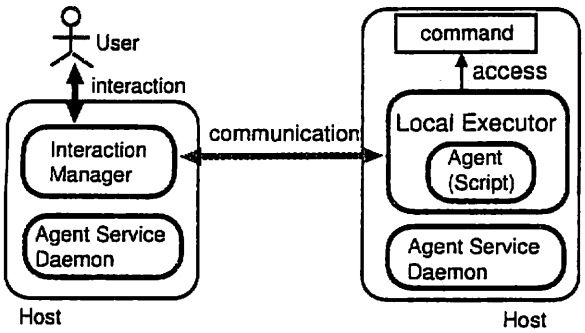

図 1　システムの概要

実行と移動を司るサーバプロセスが 1 つずつ存在する．これをエージェントサービスデーモン (Agent-Service-Daemon:AS) と呼ぶ．ユーザが実行するスクリプトは AS に渡され，AS によって移動エージェントとして実行される．エージェントが移動する場合は，AS が移動先の計算機の AS と通信を行う．

4.2　移動エージェントの実行

計算機 S 上の AS はスクリプトを受け取ると自身がスクリプトを解釈，実行するのではなく，S 上に別のプロセスを生成し，スクリプトの実行を委ねる．このプロセスをローカルエグゼキュータ (Local-Executor:LE) と呼ぶ．すなわち，スクリプトは AS とは別のプロセスとして実行されるこのため，スクリプトが何らかの理由で暴走したとしても，AS は影響を受けない．LE は，スクリプト中の各コマンドについてコマンド資源の検索と選択を行い，選択した資源が S の資源であれば実行する．遠隔計算機の資源を選択した場合，スクリプトの移動を S の AS に依頼し，自分自身は終了する．S の AS はスクリプトとスクリプトの実行状態を LE から受け取り，それを移動先の計算機 D の AS に送る．D の AS は，受け取ったスクリプトと実行状態をもとに D 上で新たに LE を生成し，スクリプトの実行を再開する．

4.3　インタラクションマネージャ

移動エージェントは自律的に動作することが一つの特徴である [1, 2, 3, 8, 9]．そのため，既存の移動エージェントシステムにおいて，他の計算機に移動したエージェントは，通常ユーザと対話することはない．一方，本システムでは，ユーザが実行したスクリプトは，ユーザからは通常のプロセスとして認識される．ユーザは通常のプロセスと同様に，実行途中のスクリプトからなんらかの出力を受け取ったり，あるいはシグナルを送って実行を中断，中止したい場合などがある．このため，本システムではインタラクションマネージャ(Interaction-Manager:IM) を導入した．IM はユーザがスクリプトを起動した計算機に留まり，ユーザと実行中のスクリプトの対話を受け持つ．

スクリプトはその移動に伴い，移動先の計算機上の LE に順にその実行が引き継がれて行く．ユーザと LE は IM を介して対話を行うため，あるスクリプトの LE と IM は常に互いの場所を把握している．

ユーザがスクリプトを実行すると，まず IM が起動され，IM が AS にスクリプトを転送する．IM はスクリプトの実行が終了するまで計算機上に存在する．LE の出力は IM を通じてユーザに返され，ユーザが IM にシグナルを送ると，IM から LE にシグナルが転送される．もし LE が異常終了した時には，IM に通知され，IM もまた終了する．このため，ユーザには IM がスクリプトを実行中のプロセスそのものであるかのように見える．これにより，ユーザはスクリプトがどの計算機で実行されているかを意識すること無く．通常のプロセスと同様に対話できる．IM，LE，AS の関係を図 1に示す．

4.4　ローカルスクリプト

既存の移動エージェントシステムでは，ユーザの記述したスクリプトは，複数の計算機間を移動しながら実行されることが前提となっている．このため起動されたスクリプトはすべて移動エージェントとして実行される．本システムは分散計算機環境におけるコマンドインタプリタであり，本システムで実行される多くのスクリプトは他の計算機のコマンド資源を利用しないスクリプトである．以降このようなスクリプトをローカルスクリプトと呼ぶ．ローカルスクリプトを実行する際に，IM から AS にスクリプトを転送し，AS があらためて LE を生成し，LE にスクリプトを実行させるのは効率が悪い．よって，本システムでは，IM に LE の機能を持たせ，IM はユーザからスクリプトを受け取ると，遠隔コマンド資源へのアクセスが発生するまでは AS に転送せず，IM 自身が LE としてスクリプトの解釈と実行を行う．これにより，ローカルスクリプトを実行する場合に LE の生成や IM と AS の間の通信といったオーバヘッドをなくすことができる．

4.5　ファイルへのアクセス

既存の移動エージェントシステムでは，通常，エージェントのオープンしたファイルに移動後にアクセスすることはできない．なぜならオープンしたファイルのファイル記述子は OS によって管理されており，エージェントが移動すると，そのエージェントは異なる計算機の OS 上で実行されることになるためである．

本システムでは，ユーザはスクリプトの実行される計算機を意識しない．このため，スクリプト中であるファイルをオープンし，遠隔コマンド資源にアクセスした後，そのファイルにアクセスするようなスクリプトを書く場合があり得る．スクリプトが移動後にファイルにアクセスできないと，このようなスクリプトを実行することができなくなる．

よって，本システムでは，LE が直接ファイルをオープンしアクセスすることはない．LE はファイルの存在する計算機の AS にファイルのオープンを依頼し，ファイルへの実際のアクセスは AS が行う．

表 3　実験結果

インタプリタ	計算機	時間 (秒)
試作システム	eiger	6.35
試作システム	k2	6.32
tcsh	eiger	10.80
tcsh	k2	6.30

LE は AS と通信を行いながら間接的にファイルにアクセスする．しかし，もし LE が異常終了した場合，AS がオープンしたファイルが，クローズされないという問題がある．そこで，LE からのファイルのオープンの依頼を受け取ると，AS は対応する IM に対し，オープンしたファイルの位置と名前を通知する．もし LE が異常終了するとそのことが IM に通知され，IM は終了する前に，AS によってオープンされたファイルがクローズされていなければ，それらをクローズするよう AS に依頼する．

5. 実験

3.5節で述べたコマンド資源の選択の有効性，スクリプトの計算機間の移動やコマンド選択のオーバーヘッドなどを調べるために，実験を行った．

次のような状況で実験を行った．あるユーザーのメールボックスの中に，電子メールが 4000 通存在する．このなかから，あるキーワードを含むメールのファイル名のリストを得たい．この仕事を行うスクリプトを Perl で記述し，本研究で実装したコマンドインタプリタと，一般的なコマンドインタプリタである tcsh 上でそれぞれ実行し，実行時間を測定した．実験を行った分散計算機環境上には，同じ構成 (CPU:PentiumII 300Mhz，メモリ 128MB，OS:Linux 2.2.18) の 2 台の計算機 k2 と eiger が存在し，メールは k2 に保存されている．また，メールは NFS により eiger からも参照することができる．これらのメールを検索するためのスクリプトは，両方の計算機上に存在する．また，この 2 つの計算機は 100Mbps のイーサネットで接続されている．実験結果を表 3で示す．表中の項目「計算機」とは，スクリプトを最初に起動した計算機である．計算機 eiger 上でスクリプトを実行した場合，実装したシステムを用いることによって，用いない場合と比べて，約 60%の時間 (約 4.5 秒差) で実行が終了している．このことから，この例では，3.5節で述べたコマンド資源の選択が有効に働いたといえる．実装したシステムを用いて，スクリプトを計算機 k2 上で実行した場合と，計算機 eiger 上で実行した場合とでは，実行時間に差がないことから，スクリプトの移動のオーバーヘッドはほとんどないと言える．また，スクリプトを計算機 k2 で実行した場合の時間差から，資源選択に要したオーバーヘッドもほとんどないと言える．これらの時間は，特に対話的な実行においては問題ない程度に短いと考えられる．

6. まとめ

本稿では，従来の分散計算機環境上での遠隔コマンドの利用における問題点を指摘し，それを解決するため，分散計算機環境上でコマンド資源に対し，位置透過かつアクセス透過にアクセスすることのできるシステムを実装した．実装したシステムは，以下の様な特徴を持つ．(1) 遠隔コマンド資源に対してアクセスするために，移動エージェントの仕組みを採用した．(2) コマンド資源の情報を管理する仕組として，コマンドのインポートとエクスポートの考え方を導入した．(3) コマンドの実行の際，コマンドが引数にとるファイルに着目し，コマンドの実行時間が短くなるようなコマンドの選択を可能にした．また，作成したシステムを用いて実験を行い，コマンド選択の効果とシステムの有用性を確認した．

今後は，あらたなコマンド資源の選択方法を実装し，実験を行って評価する予定である．また，他の分散ファイルシステム (例えば Coda) に応じた実装等を行う予定である．

References

[1] R. S. Gray : "Agent Tcl: A transportable agent system. In Proceedings of the CIKM Workshop on Intelligent Information Agents", Fourth International Conference on Information and Knowledge Management, 1995

[2] D. Chess, B. Grosof, C.Harrison, D. Levine, C. Parris, and G. Tsudik : "Itinerant Agents for Mobile Computing",IEEE Personal Communications, pp.22-49, 1995

[3] D. Johanson, R. V. Renesse, and F. B. Schneider : "An Introduction to the TACOMA Distributed System Version1.0", Technical Report 95-23, Depth. of Computer Science, Univ. of Tromosø and Cornell Univ., Tromosø, Norway, June 1995

[4] K. Pradeep and Sinha : "Distributed Operating Systems Concepts and Design", IEEE PRESS,ISBN 0-7803-1168-X

[5] Brian C. Smith, Lawrence Rowe, Stephen Yen. : "Tcl Distributed Programming", In Proceedings of the 1st Tcl/Tk Workshop . June 1993.

[6] Tanenbaum, A.S., Kaashoek, M.F., Renesse, R. van, and Bal, H.: "The Amoeba Distributed Operating System-A Status Report,", Computer Communications, vol. 14, pp. 324-335, July/August 1991.

[7] M. Satyanarayanan et al., Coda: "A Highly Available File System for aDistributed Workstation Environment", IEEE Trans. on Comp., vol. 39, no. 4, Apr. 1990.

[8] H. Peine : "An Introduction to Mobile Agent Programing and the Ara System", ZRI-Report, january 1997

[9] M. Straßer, J. Baumann, and F. Hohl : "Mole - A Java Based Mobile Agent System", Object-Oriented Programming ECOOP'96, M. Mühlaüser, ed., pp. 327-334, july 1996