卒業論文

グリッド計算システム AgentTeamwork における
入出力機能とユーザインタフェースの開発に関する研究

愛媛大学 工学部 情報工学科

平成 14 年度入学

宮内 順平

指導教員

小林 真也 教授

平成 18 年 2 月 15 日提出

概論

高速ネットワークの普及は、計算機のネットワーク常時接続という環境をユーザに与えた。グリッド計算は、これらの計算機を有効利用し、複数の計算機で並列に処理を行うことで、仮想的に高性能な計算機を作り出すというものである。AgentTeamwork は、モバイルエージェント（異なる計算機間を、状態を保持したまま自律的に移動するプログラム）を利用したグリッド計算システムである。AgentTeamwork のモバイルエージェントは、グリッド環境でユーザのジョブを実行するために、遠隔の計算機に移動する。移動したエージェントは、エージェント間で木構造を形成し、移動先の計算機上でユーザのジョブを並列に実行し、実行結果をユーザのもとに返送する。現在の AgentTeamwork の実装には、入出力機能、およびグラフィカルユーザインタフェースが含まれていない。このため、ユーザはジョブの投入時に長いコマンドを打つ必要があり、さらに、入出力データを扱うジョブを実行することができないという制限を持っていた。そこで本研究では、AgentTeamwork の入出力機能を開発することで、エージェントの木構造を利用した入力データの転送、および GridFile の提案による入出力データの排他制御を実現した。また、グラフィカルユーザインタフェースを開発することで、容易なジョブの投入、および各ユーザプログラムからの出力データの個別表示を実現した。本論文では、これらの実装方法について述べる。また、本研究で開発したこれらの機能を、既存のグリッド計算システムの入出力機能、およびユーザインタフェースと比較し、その優位性について検証する。

目次

第 1 章　緒論 1

第 2 章　AgentTeamwork 3

2.1　概要 3

2.2　エージェントの構成 4

2.3　AgentTeamwork の入出力機能、およびユーザインタフェースの現状 6

第 3 章　入出力機能の開発 9

3.1　設計方針 9

3.2　実装 10

3.2.1　入出力ファイル、および標準入出力データの転送 10

3.2.2　リフレクション 12

3.2.3　GridFile 13

3.3　測定・評価 18

第 4 章　ユーザインタフェースの開発 21

4.1　設計方針 21

4.2　実装 21

4.3　機能に関する考察 25

第 5 章　関連研究 27

5.1　入出力機能 27

5.2　ユーザインタフェース 29

5.3　AgentTeamwork との比較 31

第 6 章　結論 **35**

謝辞 **37**

参考文献 **39**

第 1 章　緒論

近年、ISDN、DSL、ケーブルテレビネットワークの普及にともない、多くの一般ユーザが所有する計算機は、常時、ネットワークへ接続されることになった。また、教育機関や商用機関では、常時接続を利用したクラスタシステムが定着した。しかしながら、これらの計算機は、常にユーザによって使用されているわけではなく、教育機関、および商用機関では約 70%の計算機が利用されていない状態でネットワークに接続されている[3]。グリッド計算システムは、これらのネットワーク上の利用されていない軽負荷な計算機群を利用して、仮想的に高性能な計算機を作り出すシステムである。このシステムを利用することにより、性能の低い計算機からでも、ネットワークが利用可能な状態であれば、高速に、かつ大量の処理を行うことが可能となる。

我々は、モバイルエージェント[13]を利用した、グリッド計算システム AgentTeamwork プロジェクト[1][2]の研究を行っている。モバイルエージェントとは、異なる計算機間を、状態を保持したまま自律的に移動するプログラムである。 AgentTeamwork は、このモバイルエージェントを利用し、ユーザのジョブを遠隔の計算機上で実行し、その結果をユーザのもとへ返送する。

現段階では、AgentTeamwork は、入出力を扱うユーザジョブを実行できないという制限を持つ。また、現在の AgentTeamwork では、ユーザが行う処理は、すべてコマンドラインから行う必要がある。よって、ユーザは、ユーザジョブを実行する計算機資源を要求するために、長いシェルコマンドを、コマンドラインから打たなければならない。これらの問題を解決するため、本研究では、AgentTeamwork の木構造を利用した入出力機能の開発、および、Java アプレット[4]を使用したグラフィカルユーザインタフェースの開発を行った。入出力機能では、入出力の転送に木構造を使用することで、NFS より効率的な転送が行えることが分かった。また、インタフェースでは、アプレットを使用することにより AgentTeamwork の、サーバを介さないという特徴を損なうことなくインタフェースの開発が行えることが分かった。本論文では、これらの機能の実装、および有効性について述べる。

本論文の構成は以下の通りである。第 2 章では、 AgentTeamwork の概要について紹介し、現在の実装における問題点について言及する。第 3 章では、AgentTeamwork の特徴を利用した独自の入出力機能を提案し、その機能を実装することによって実現した、ファイル入出力、および標準入出力の実装技術について述べる。第 4 章では、AgentTeamwork インタフェースを実装することによって実現した、ユーザとユーザプログラム間の入出力、およびジョブ投入の実装方法について述べる。第 5 章では、開発した入出力機能、およびユーザインタフェースを、既存のグリッド計算システムの入出力機能、およびユーザインタフェースと比較し、その機能の違いを検証する。最後に、第 6 章で本論文の結論を述べる。

第 2 章　AgentTeamwork

本章では、AgentTeamwork の概要および特徴について述べる。そして、入出力、およびユーザインタフェースの実装においての問題点を考察し、その問題の解決方法について言及する。

2.1　概要

AgentTeamwork は、モバイルエージェントを利用した Java ベースのグリッド計算システムである。このシステムは、各計算機のリソース情報を登録するために FTP サーバを利用している点を除いて、すべての計算機管理は、分散方式に基づいて行われる。

AgentTeamwork は、4 種類のエージェントを用いる。これらのエージェントは、ユーザのジョブとともに遠隔の計算機へ移動し、これを実行した後、その実行結果をユーザの元へ返送する。図 2.1 は、これらのエージェントによる、ネットワーク上での処理の流れを示したものである。図中の数字は、以下の数字と対応している。

(1) ユーザは自分の端末からジョブを投入する。ジョブが投入されると、コマンダエージェントと呼ばれるエージェントが生成される。コマンダエージェントは、ジョブの移動先を決定するリソースエージェントを生成する。

(2) リソースエージェントは、ローカルのリソースデータベース（および必要ならば共有サーバ）に問い合わせて、計算機情報が記述された XML ファイルを検索し、ユーザの要求に適する計算機を決定する。

(3) コマンダエージェントは、ジョブの実行を担うセンチネルエージェントと計算の途中結果を保存するブックキーパエージェントを生成し、決定された計算機群に送る。

(4) 遠隔の計算機に送られたセンチネルエージェントは、そこでユーザジョブを実行する。

(5) センチエネルエージェントは、現在実行している計算機の障害に備えて、ジョブの実行結果を、関数の実行毎にブックキーパエージェントに送る。

(6) ユーザジョブの実行を終えたセンチネルエージェントは、その実行結果をユーザの下へ返す。

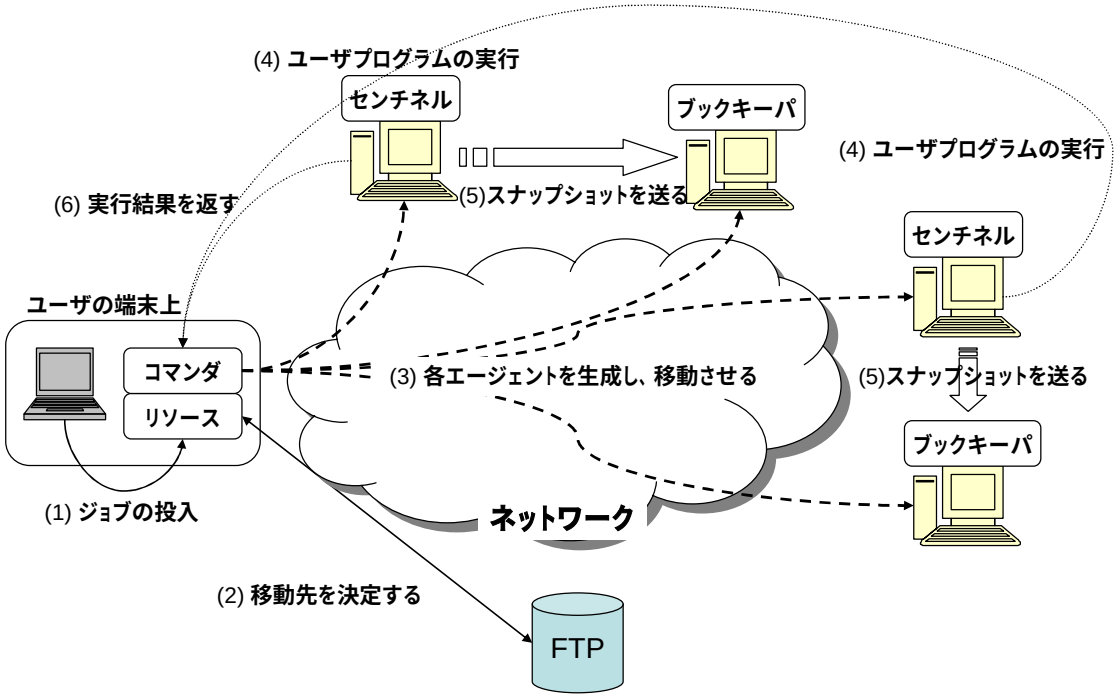

図 2.1 AgentTeamwork のネットワーク上でのエージェント配置

2.2 エージェントの構成

AgentTeamwork における、すべてのエージェントは、UWAgent 抽象クラスを拡張して生成される。UWAgent を拡張した各エージェントは、自分の子供を生成することができる。生成できる子供の最大値を N とした場合、UWAgent は、式（1.1）にしたがって、生成した自分の子供に *id* を割り振る。式中の i は、子供を生成するエージェントの *id* であり、0 から始まる。

$$id = i * N + seq \quad \begin{cases} 1 \le seq \le N-1 & (i = 0) \\ 0 \le seq \le N-1 & (i \neq 0) \end{cases} \tag{1.1}$$

この式を用いることにより、各エージェントは、自分の子供を生成し、ネームサーバを介すことなく *id* を割り振ることができる。これらのエージェントは、親子の関係となり、図 2.2 に示されるような木構造を形成する。この図では、子供を生成できる最大値を 4 と仮定してあるため、各エージェントは 4 人まで子供を生成する。ユーザの要求した計算機数に満たない場合は、生成された子供がさらに子を生成することでその要求を満たす。

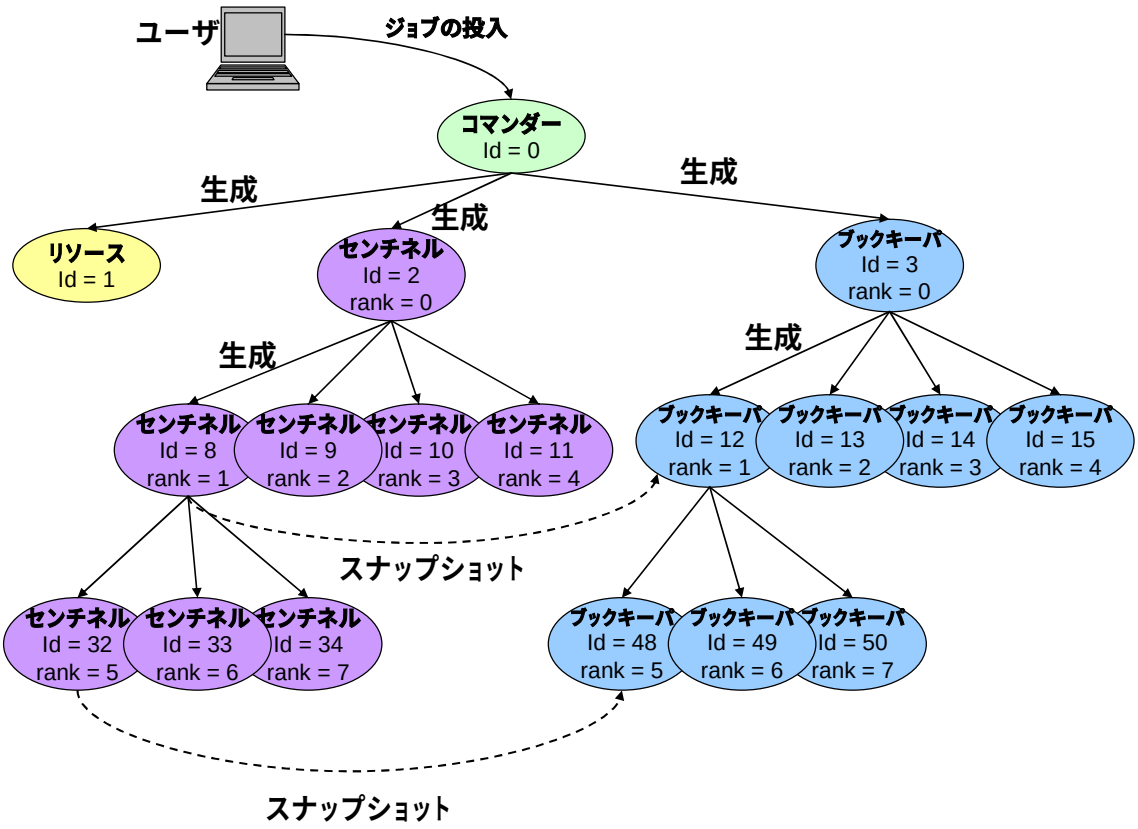

図 2.2　AgentTeamwork におけるエージェント構成

また、図 2.2 に示されるように、センチネルエージェントとブックキーパエージェントは、その *id* から計算される MPI ランクを持つ。センチネルエージェントは、スナップショットを送る場合、自分のランクに対応するブックキーパエージェントにスナップショットを送

る。このため、同じランクのセンチネルエージェントとブックキーパエージェントは、対の関係となる。

AgentTeamwork におけるエージェント間通信は、エージェント *id* と、エージェントの木を使用して行われる。エージェント間でメッセージの交換を行う場合、メッセージは、この木の枝をたどっていきながら目的のエージェントに届けられる。しかし、通信をするたびに枝をたどるのは非効率であるため、AgentTeamwork では、エージェント間通信の性能改善のために、一度通信をしたエージェントの IP をキャッシュすることで、次に通信する際には直接目的のエージェントと通信することができる。

2.3 AgentTeamwork の入出力機能、およびユーザインタフェースの現状

現在の AgentTeamwork は、入出力に関して 4 つの制限がある。以下にそれらを列挙する。

- 遠隔ノード上で実行されているユーザプログラムの標準出力は、ユーザのローカルノードまで届かない。
- ユーザプログラムへの標準入力がおこなえない。
- データファイルを扱うユーザプログラムを実行することができない。
- ユーザプログラムが出力したファイルは、ユーザの下へ届けられない。

これらの制限のため、ユーザは標準入出力やファイルを扱ったプログラムを実行することができない。次章では、この標準入出力、およびファイル入出力を実現するため、GridFile というクラスを作成し、その機能について述べる。GridFile の開発により、ユーザは、AgentTeamwork を通して、標準入出力、入力データファイルを扱ったプログラムの実行、出力ファイルのユーザへの返送が可能となる。

また、現在の AgentTeamwork には、グラフィカルユーザインタフェースを含んでいない。このため、ユーザは、ジョブの投入時に、コマンドラインから、計算機リソース要求

のための長いシェルコマンドを打つか、あるいはそれに相当するシェルスクリプトを作成する必要がある。また、入出力機能の追加にともなって、遠隔の各ユーザプログラムへの標準入力、および各ユーザプログラムからの標準出力を表示するためのインタフェースが必要となる。これらの要求を満たすため、簡単なグラフィカル操作のみでジョブの投入が可能になるユーザインタフェースを開発する。このインタフェースを開発することで、ユーザは画面操作にてジョブを投入することができ、さらに各ユーザプログラムの標準入出力を管理できるようになる。

第 3 章　入出力機能の開発

本章では、AgentTeamwork の入出力機能の開発について述べる。3.1 節では、その設計方針を明白にし、3.2 節では、その設計方針にもとづいた、入出力機能の実装について述べる。3.3 節では、実装した AgentTeamwork の入出力機能を、NFS と比較し、その有効性について検証する。

3.1　設計方針

AgentTeamwork の入出力は、ファイル入出力、および標準入出力の 2 種類である。これらのデータは、ユーザが意識することなく、ユーザから遠隔ノードで実行中の各ユーザプログラムへ、または、各ユーザプログラムからユーザのもとへ届けられなければならない。この機能を実現するために、本研究では、3 つの設計方針を決定した。

1. 入出力ファイル、および標準入出力データの転送
   必要な入力ファイル、および標準入力データは、コマンダエージェントから、エージェントの木を使用して、並列にユーザプログラムを実行する計算機に転送する。また、出力ファイル、および標準出力データは、ユーザプログラムから出力されるたびに、直接コマンダエージェントに返送する。

2. リフレクション
   入出力ファイルは、センチネルエージェントがユーザジョブを実行する計算機上のメモリ、または、/tmp 内の特定のディレクトリに格納する。後者の場合、ユーザプログラムは、そのディレクトリの位置を動的に知る必要がある。本研究で開発する入出力機能では、リフレクションと呼ばれる、クラスからフィールド変数を取得できる Java API を使用することでこの要求を満たす。

3. GridFile

遠隔ノード上での入出力ファイル、および標準入出力データは、（１）入力ファイル、および標準入力データを扱うスレッド、（２）出力ファイル、および標準出力データを扱うスレッド、そして（３）ユーザプログラムを実行するスレッド、の 3 つで共有される。このため、これらのデータは排他的に制御される必要がある。この排他制御を実現し、スレッド間での整合性を保ったファイル、およびデータアクセスを行うため、それらのスレッドと共有データを仲介する GridFile というクラスを作成する。

3.2 実装

本節では、3 つの設計方針にもとづいた、入出力機能の実装について記述する。この機能により、AgentTeamwork は、ファイル入出力、および標準入出力を必要とするユーザジョブを遠隔の計算機上で実行することが可能となり、入出力を必要とするユーザプログラムを実行することができるようになる。

3.2.1 入出力ファイル、および標準入出力データの転送

以下に、（１）ユーザプログラムへの入力ファイル、および標準入力データの転送方法、および、（２）ユーザへの出力ファイル、および標準出力データの返送方法について述べる。

(1) 入力ファイル、および標準入力データ

図 3.1 に、入力ファイルの転送方法を示す。AgentTeamwork の入力ファイル転送は、ユーザ側から一つ一つファイルを指定するのではなく、ファイルがおいてあるディレクトリを指定する。このようにすることで、入力ファイルが複数の場合にも柔軟に対応することができる。入力ファイルは、図 3.1 に示されるように、指定したローカルホスト上のディレクトリにコマンダエージェント自身が読みに行く。

また、入力ファイルを、一定のブロックに分割し、各センチネルエージェントへ転送することにより、その転送効率を上げる。コマンダエージェントは、バイト型の入力ファイルデータを、ブロックサイズ分だけ読み込み、ファイル名をキーとして

ハッシュテーブルに格納し、これをエージェントの木を使って送信する。この転送方法により、エージェント数の増加にともなう、一つの計算機あたりにかかる通信負荷は、最大でも、自分の子供の数に限定される。また、標準入力データも同様に木を使って送信され、その入力データを受信した各センチネルエージェントは、データに付加されたヘッダ情報から、自分に該当するデータであるかどうか調べる。自分に該当するデータであるならば、それを受け取り、それ以下のエージェントには送信をしない。これにより、無駄な入力データの送信を防ぐことができる。

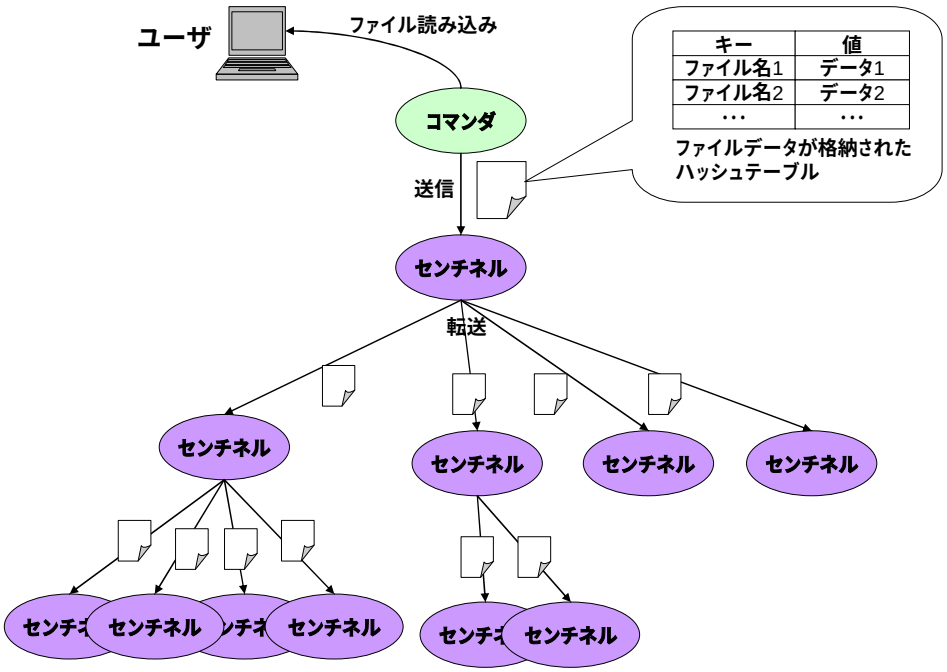

図 3.1:　入力ファイル転送

(2) 出力ファイル

出力ファイルは、ユーザプログラムの終了によらず、出力データが生成された時点でユーザのもとへ送られる。この方法を用いることにより、ユーザの計算機は、ユーザプログラムの実行終了にともなった出力ファイルの一斉受信を分散することができる。これを実現するには、センチネルエージェントは、遠隔の計算機上で、入力ファイルと出力ファイルを識別する必要がある。よって、AgentTeamwork では、出力ファイルを以下のように定義する。

出力ファイル：出力ファイルは、ユーザプログラムによって新しく生成されたファイル、あるいはユーザプログラムによって入力ファイルに新たにデータが追加されたファイルである。

この定義により、センチネルエージェントは、ユーザプログラムから操作されたファイルを検出することができるようになり、出力ファイルを検出すると、直ちにそのファイルをユーザのもとへ返送する。ユーザのもとへ返送されてきた出力ファイルは、ジョブの投入時に指定した、入力ファイル用のディレクトリに格納される。また、標準出力データも同様に、ユーザプログラムから出力されると、直ちにセンチネルエージェントは、そのデータをユーザのもとへ返送し、ユーザインタフェース上に表示する。

以上の方法により、入力ファイルは、ユーザが指定したディレクトリから自動で読み込まれ、転送される。出力ファイルは、出力に応じて自動的に返送され、そのディレクトリへ書き込まれる。ユーザは、ジョブの投入から終了にいたるまで、全くファイルの動きを意識する必要が無く、ファイルの透過性が実現されている。

3.2.2 リフレクション

センチネルエージェントが、入力ファイルを格納するために生成するディレクトリ名は、計算機が有する IP、タイムスタンプ、そしてエージェント *id* によって識別される。これにより、異なるエージェントが同一の計算機でジョブを実行した場合でも、各エージェントは自分が受け持つ入力ファイルを区別することができる。しかしながら、この区別を行うためには、ユーザプログラムは動的にそのディレクトリ名を知る必要がある。AgentTeamwork では、ユーザプログラムにこのディレクトリ名を通知するために、リフレクションを使用する。

リフレクションは、クラスからフィールド変数を取得できる API である。図 3.2 を用いて、リフレクションにもとづく、ユーザプログラムへのディレクトリの通知方法の手順を以下に示す。

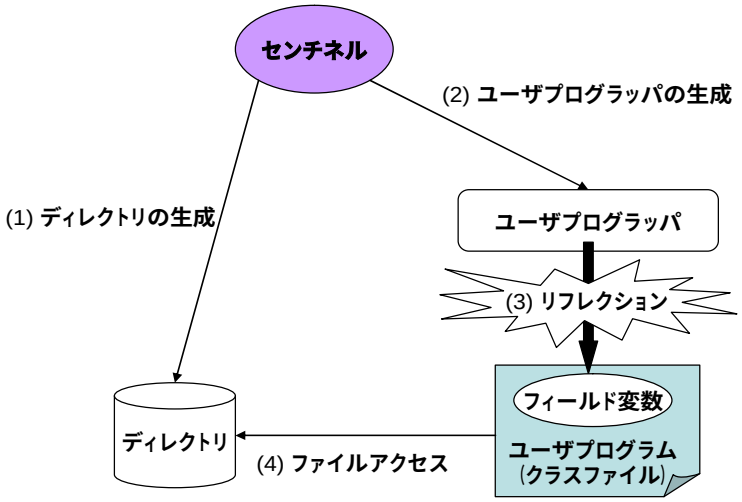

図 3.2:　リフレクションを使用したディレクトリの通知方法

(1) IP、タイムスタンプ、エージェント *id* を用いてディレクトリを生成する。

(2) 実際にユーザプログラムを実行するユーザプログラッパオブジェクトを生成する。

(3) リフレクションを用いて、ユーザプログラムのフィールド変数を取得する。このフィールド変数を使って、ユーザプログラムのデータ変数にディレクトリの位置を設定する。

(4) ユーザプログラムが開始される時点で、以上の操作が終わっているため、ユーザプログラムはディレクトリの位置を自身のデータ変数から知ることができる。よって、センチネルエージェントが動的に生成したディレクトリへのアクセスが可能となる。

3.2.3　GridFile

図 3.3 は、センチネルエージェントが子スレッドを生成し、これらを使って入出力を制御する様子を示している。センチネルエージェントは、遠隔ノードに移動した後、自分がすべき処理を行うためにいくつかのサブスレッドを生成し、マルチスレッドとして動作する。

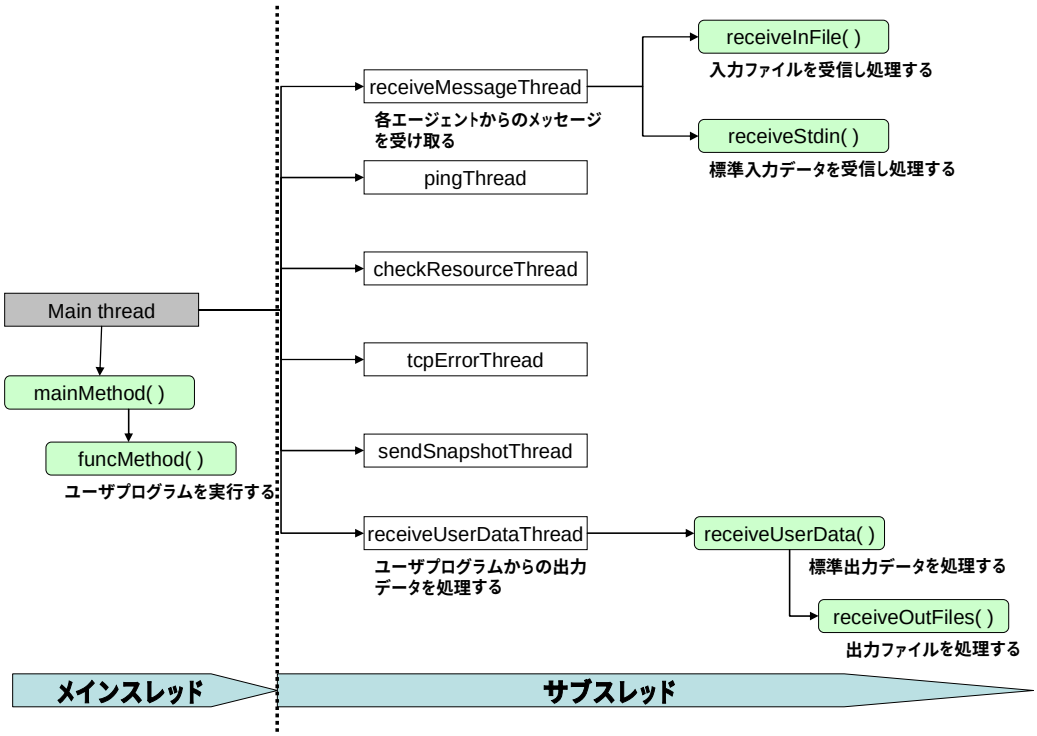

図 3.3:　センチネルエージェントのスレッド構成

ユーザプログラムを実行するスレッドはメインスレッドである。入力ファイル、および、標準入力を受信し、処理するのは receiveMessageThread（別名、入力スレッド）、そして、出力ファイル、および標準出力を処理するのは receiveUserDataThread（別名、出力スレッド）である。これら 3 つのスレッド間で共有される入出力データは、GridFile と呼ばれるオブジェクトに格納される。メインスレッドで実行されるユーザプログラムは、GridFile オブジェクトを直接操作するのではなく、入力データには GridFileInputStream オブジェクト、出力データには GridFileOutputStream オブジェクトを使用する。これらは、GridFile を包んだオブジェクトであり、ユーザが、入出力データを扱うプログラムを組む際に、GridFile 内部の操作を隠蔽する役割を担う。このオブジェクトを使用することで、ユーザは、Java の、System.in、FileInputStream、System.out、および FileOutputStream オブジェクトと同様の操作で、AgentTeamowrk 上の入出力データを扱うことができる。　入力ファイルは、ファイルサイズが、ブロックサイズよりも大きければ分割されて転送されるため、必ずしもユーザのプログラムが開始される前にすべての入力ファイルの転送

が完了しているわけではない。また、標準入出力は、頻繁にアクセスされる。したがって、入出力データの整合性を保つために、GridFile において、これらのスレッド間の同期をとる必要がある。以下、同期方法を中心に、スレッド間の入出力データの交換について述べる。

- 標準入出力

  図 3.4 は、GridFile を用いた 3 つのスレッド間の標準入出力処理を示したものである。コマンダエージェントから、エージェントの木を介して送られてきた標準入力データは、入力スレッドによって受信され、直に GridFile 内の stdin キューに格納される。ユーザプログラムを実行するメインスレッドは、必要に応じてデータを取り出す。

  一方、ユーザプログラムが生成した標準出力データは、メインスレッドによって GridFile 内の stdout キューに格納される。出力スレッドは、stdout キューを常に監視し、新たな標準出力データを取り出し、コマンダエージェントに直接返送する。これら 2 つのキューは、Java の Vector オブジェクトとして生成され、synchronized ブロックを使って標準出力データの一貫性を保っている。

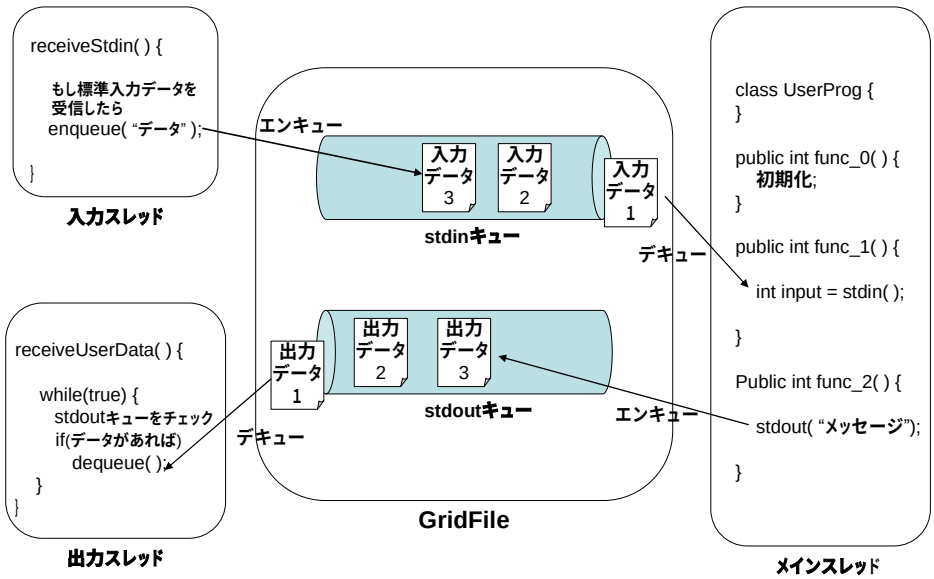

図 3.4: 標準入出力における GridFile の処理

- **ファイル入出力**

  GridFile はその内部でユーザの入出力ファイルを管理するが、特にこれらのファイルは、整合性を保つために、以下の排他制御を行う必要がある。

  1. ファイルブロックに同時にアクセスされてはいけない
  2. 誰かが書き込みをしているファイルは、その書込み処理が終了するまで書き込みをすることは出来ない

  上述の排他制御を行うため、GridFile は各ファイルに、“ オーナー ”と“ モード ”という 2 つの属性をファイルに割り当てる。これらの属性は、以下のように定義される。

  - **オーナー**

    GridFile のオーナー属性とは、そのファイルを操作できるスレッドを示す。

  - **モード**

    GridFile のモード属性とは、ファイルの操作状況を表す。このモードには、“ null ”、“ write ”、そして“ finished ”の 3 種類がある。“ null ”は、そのファイルがまだ存在していない状態を表す。“ write ”は、そのファイルに現在書き込み処理が行われている状態を表す。“ finished ”は、そのファイルの書込み処理が終了している状態を表す。

  GridFile の基本的なファイル操作は、（1）ファイルのオープン、（2）ファイルのクローズ、（3）読み込み操作、（4）書き込み操作、の 4 つである。以下に、これらの操作を、GridFile のメソッドを用いて説明する。

  (1) open(String filename, String op) ：ファイルをオープンする

  GridFile におけるファイルのオープンには、読み込みオープン（op=“ R ”）と、書込みオープン（op=“ W ”）の 2 種類が存在する。

  読み込みにおけるファイルオープンは、ファイルの読み込みが可能な状態にフ

ァイルを初期化する。ただし、ファイルモードが“ null ”である場合は、その指定したファイルは親のエージェントからまだ転送されてきていないということを意味するため、このファイルをオープンしたいスレッドは、そのファイルが生成されて、ファイルモードが“ write ”になるまで待たなければならない。
一方、書込みにおけるファイルオープンは、そのファイルのオーナーを自分に設定し、そのファイルのモードを“ write ”に変更する。もしオープンしたいファイルのモードが、すでに“ write ”だった場合、他のスレッドがこのファイルに書き込みを行っているということを意味する。よって、整合性を保つために、そのファイルモードが“ finished ”になるのを待ち、ファイルのオーナーを自分に設定して書き込み可能な状態に初期化する。

(2) close(String filename)：ファイルをクローズする
このファイルが、読み込みでオープンされていた場合、そのファイルのファイルポインタを－１にセットする。このファイルが書込みでオープンされていた場合、そのファイルモードを“ finished ”に変更し、そのファイルを待っているスレッドにその所有権を譲る。

(3) read(String filename, byte[] data, int off)：データを読み込む
引数に指定した data 配列に、ファイルのデータを格納する。ディレクトリが指定されている場合には、すべてのデータはディスクから読み込まれる。ディレクトリが指定されていないなら、メモリからそのデータが読み込まれる。両者とも、読むべきファイルデータがない場合は、－１を返す。ファイルの読み手は、この返り値と、このファイルに割り当てられたモード属性を調べることで、そのファイルを完全に読み終えたのか、それともまだデータが追加される可能性があるのかどうかを知ることができる。

(4) write(String filename, byte[] data) ：ファイルにデータを書き込む
この操作は、ディレクトリが指定されていればそのディレクトリに、指定されていなければメモリ上に、そのデータをキャッシュする。データを書き終えると、このファイルを読むために待っているスレッドのために、notify メソッドを用いてそれらのスレッドがこのファイルにアクセスできるように後処理を行う。

3.3　測定・評価

本節では、AgentTeamwork の入出力機能を用いたファイルの転送を行い、その転送にかかる時間を測定する。また、NFS を使用した同様の実験を行い、AgentTeamwork の入出力機能の有効性について検証する。

測定に使用した計算機環境を、表 3.1 に示す。本測定では、DELL 計算機で構成されたギガイーサネットクラスタを用い、マスタノードから、同時に 24 台のスレーブノードへファイルを転送し、各ノードからの転送完了メッセージがすべて受信されるまでの時間を測定した。比較対象の NFS では、mpiJava[12]を使ってマスタ・スレーブ型のテストプログラムを作成し、マスタノードから各スレーブノードへ読み込み開始メッセージを送信してから、各スレーブノードからの読み込み完了メッセージをすべて受信するまでの時間を測定した。なお、ファイル転送に使用するファイルのサイズは、8M バイト、16M バイト、32M バイト、64M バイト、128M バイト、そして 256M バイトである。

表 3.1:　計算機環境

	3.2GHz / 1MB cache Xeon
OS	Linux
メモリ	512MB
HDD	36GB SCSI
大域幅	1Gbps

図 3.6 に、ファイルサイズの増加にともなう、ファイルの転送時間の推移を示す。

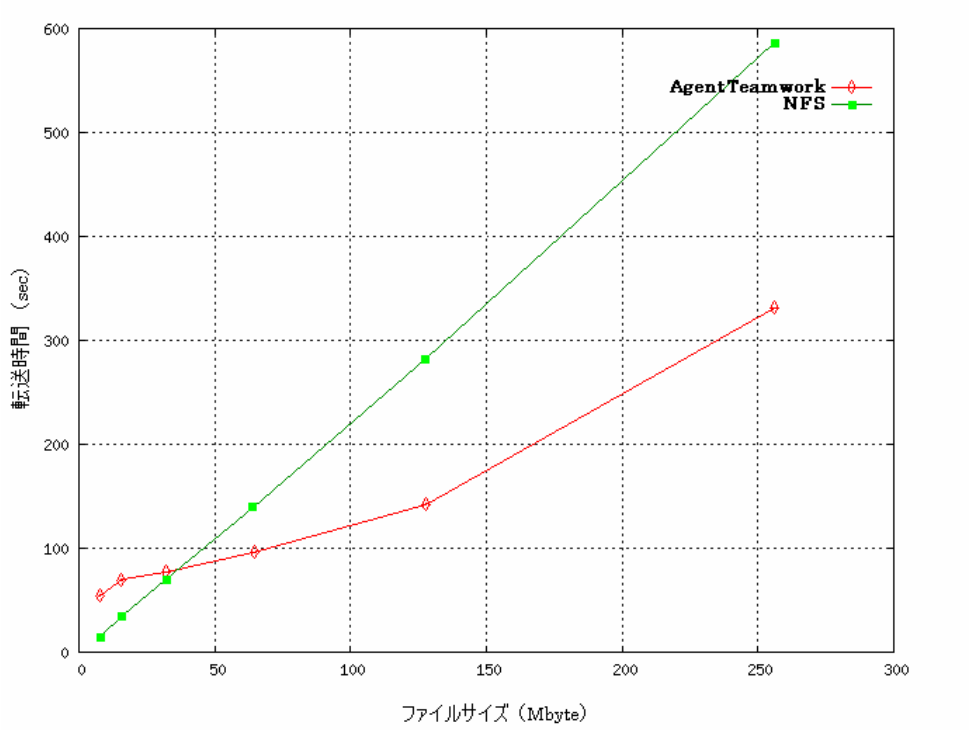

図 3.6: AgentTeamwork と NFS によるファイル転送比較

AgentTeamwork のファイル転送時間は、NFS のファイル転送時間と比較して、より効率的であることが分かる。この差は、ファイルサイズが増加するにつれて、顕著に現れている。これは、AgentTeamwork のファイル転送が、エージェントの木を利用した、ファイルサーバの負荷分散を行っているためである。AgentTeamwork では、ファイルをファイルサーバから読み込むのは、木の根となるコマンダエージェントだけである。これはすなわち、ファイルサーバにかかる負担が、24 分の 1 であるということを意味する。NFS では、同時に同じファイルサーバに多数の転送要求が到達するために、ファイルサーバに多大な負担がかかる。その負担は、転送するファイルの容量が増加するにつれて、さらに増加する。

図 3.7 は、AgentTeamwork のファイル転送において、ファイルのサイズを 2 分割した場合と、4 分割した場合における転送時間の変化を測定したものである。

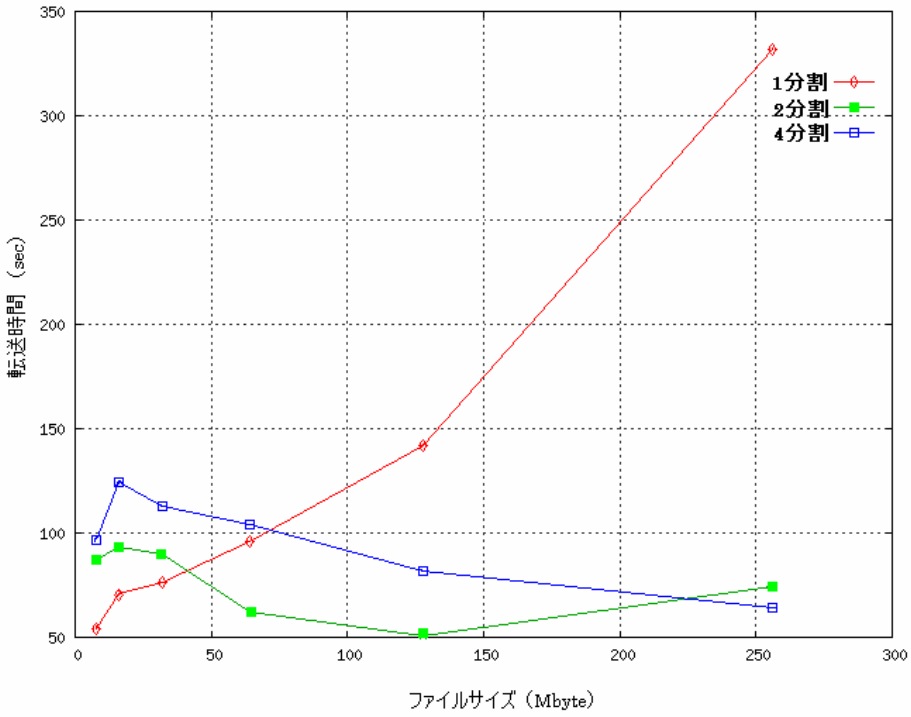

図 3.7:　分割によるファイル転送時間の比較

大容量のファイルは、分割することで、さらに効率的な転送が可能なことを確認した。256M バイトのファイル転送に関しては、2 分割よりも、4 分割の方が短時間でファイルの転送を完了できることが分かった。これは、大きな容量のファイルは、ファイルサーバからの読み込みに時間がかかるため、分割転送により、ファイルの読み込みと転送を多重化できるからである。

以上の測定より、AgentTeamwork のファイル転送は、ファイルの容量が増加するにしたがって効率が上がる。また、AgentTeamwork のファイル転送は、効率的なファイル分割数を見つけることで、256M バイトのファイルサイズにおいては、NFS による 1 対多の通信と比較して、約 7 倍の転送速度の差がでることが分かった。よって、本研究で開発した AgentTeamwork の、エージェントの木を使用したファイル転送は、非常に有効性の高いものであるといえる。

第 4 章　ユーザインタフェースの開発

本章では、アプレットを使った AgentTeamwork のユーザインタフェースの開発について述べる。4.1 節では、その開発の設計方針を明白にし、4.2 節では、設計方針にしたがって実装したユーザインタフェースについて述べる。そして、4.3 節では、実装したユーザインタフェースに関して、実装前と実装後での利便性の違いについて述べる。

4.1　設計方針

AgentTeamwork は、FTP サーバを除いて、サーバを介さないグリッド計算システムである。このため、ユーザは、自分のノードから、直接遠隔のノードにジョブを投入する必要がある。この要求を満たすため、AgentTeamwork のインタフェースには、Java アプレットを使用する。アプレットを使用することにより、ユーザは、インターネットが使える環境であれば、ブラウザ上から簡単に AgentTeamwork のシステムが使用可能になり、容易にジョブを投入することができる。AgentTeamwork インタフェースの処理は、（1）ユーザジョブを実行するための計算機資源の選択、（2）実際にジョブを投入するためのシェルスクリプトファイルの生成・実行、（3）標準入出力データを受信、送信するための、コマンダエージェントとの通信、（4）受信した標準出力データをインタフェース上で表示するための処理、および、インタフェースからの標準入力データを対応するユーザプログラムへ届けるための処理、の 4 つがある。AgentTeamwork インタフェースは、以上 4 つの処理を行うことで、ジョブの投入から実行結果を得るまでの一連の流れをユーザに提供する。

4.2　実装

ここでは、上述した設計方針に従って AgentTeamwork のユーザインタフェースを実装する。このインタフェースを実装することによって、ユーザは自分のジョブを実行する計

算機のリソースを容易に要求できるようになり、各ユーザプログラムからの標準出力データを、個別の画面を使って確認することができるようになる。さらに、必要ならば、ユーザは、遠隔ノードで実行されている各ユーザプログラムへの標準入力データの転送を行うことができる。

(1) 計算機資源の選択

ユーザは、本研究で開発したインタフェースを通して、計算機資源の選択、または入力という簡単な操作のみでジョブを投入することができる。図 4.1 は、ユーザが要求する計算機資源の選択画面である。現在選択できる資源の種類は、CPU、OS、メモリ、ディスクサイズ、CPU の数、CPU 利用率、大域幅、ユーザジョブを実行する計算機の総数、ユーザプログラム、プログラムの開始時間、そしてユーザプログラムを実行する計算機名である。ユーザは、この画面から、自分のジョブを実行させるために必要な計算機の資源をインタフェースに要求する。

(2) シェルスクリプトの生成・実行

ユーザが計算機資源を選択し、投入ボタンが押されると、AgentTeamwork インタフェースは、その要求された計算機資源をもとに、ジョブを投入するためのシェルスクリプトファイルを生成する。生成されたシェルスクリプトファイルは、Runtime クラスの exec メソッドを使用して、独立した子プロセスとして実行される。AgentTeamwork インタフェースは、子プロセスの実行出力をインタフェース上に表示するために、子プロセスから標準出力ストリームを取得し、出力する stdOutThread スレッドと、標準エラーストリームを取得し、出力する errOutThread スレッドを生成する。これらのスレッドは、出力があるたびに、インタフェース上のテキストエリアにその文字を追加していく。

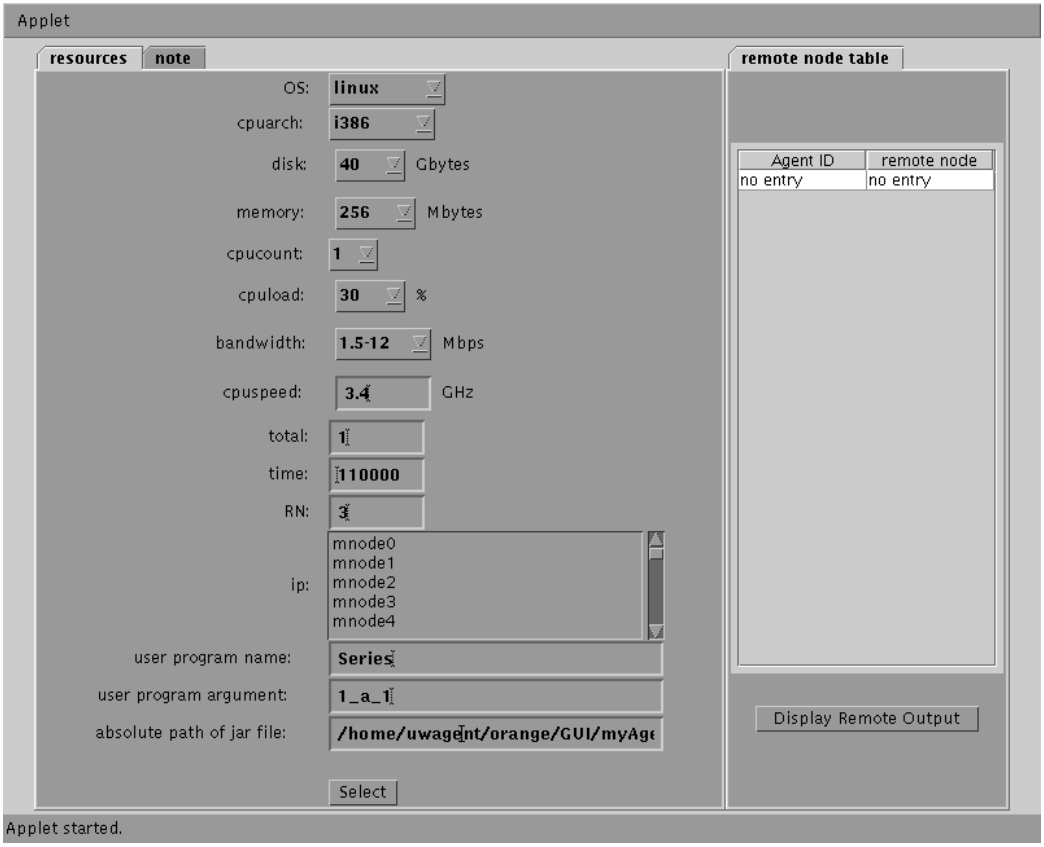

図 4.1: 計算機資源の選択画面

(3) **コマンダエージェントとの通信**

AgentTeamwork **の入出力機能では、各ユーザプログラムからの標準出力は、コマンダエージェントにすべて送られる。**AgentTeamwork **インタフェースは、そのデータを表示するために、コマンダエージェントからのソケット通信を受け付け、それらのデータを受信する**

AgentTeamwork **インタフェースは、このソケット接続の受け付けを、新たなスレッドに行わせる。このスレッドは、コマンダエージェントからの接続要求だけに対応し、実際のデータの受け付けは、子スレッドに行わせる。**

(4) **標準入出力データの処理**

図 4.2 **は、各センチネルエージェントから** AgentTeamwork **ユーザインタフェースまでの標準出力データの流れを示したものである。**AgentTeamwork **ユーザインタ**

フェースは、各ユーザプログラムから送られてきた標準出力データを、ユーザプロセスごとに異なる画面に出力する。これを実現するために、コマンダエージェントは、各センチネルエージェントから送られてきた標準出力データに、送信したエージェントの *id*、および送信元の計算機のホスト名から成るヘッダ情報を付加する。ユーザインタフェース側では、送られてきた標準出力データのヘッダ情報を調べることで、どのユーザプログラムから送られてきたデータであるか判別することができる。

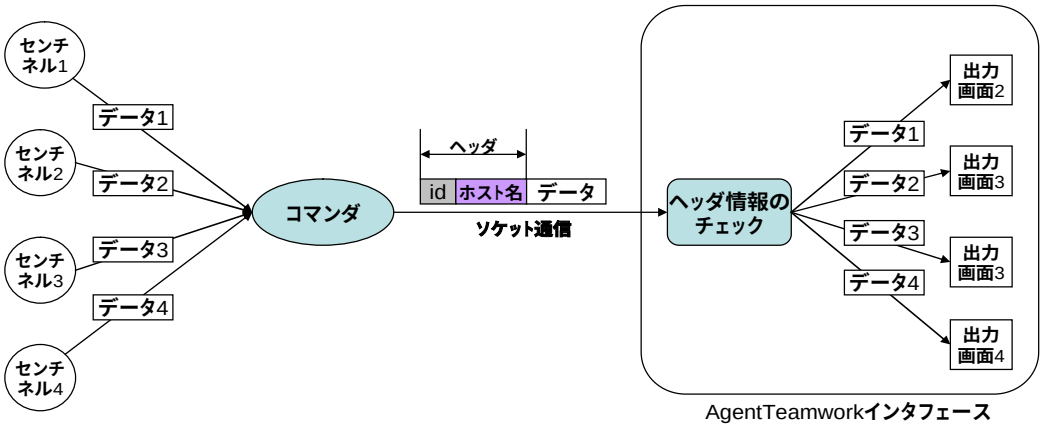

図 4.2:　標準出力データの判別方法

図 4.3 は、各ユーザプログラムから送られてきた標準出力を表示する画面である。図の左側は、標準出力をユーザプログラムごとに表示している。図の右側は、遠隔ノードへ移動したセンチネルエージェントの *id* と、移動した計算機の IP との対応表である。この表の項目は、標準出力データが送信されてきた順番に追加される。

また、標準入力は、各ユーザプログラムの標準出力を表示する画面上のテキストフィールドに書き込むことで、対応するユーザプログラムへ届けられる。この場合も、送信する標準入力データに、送信先のエージェント *id* を付加することで、目的のエージェントに届けることができる。

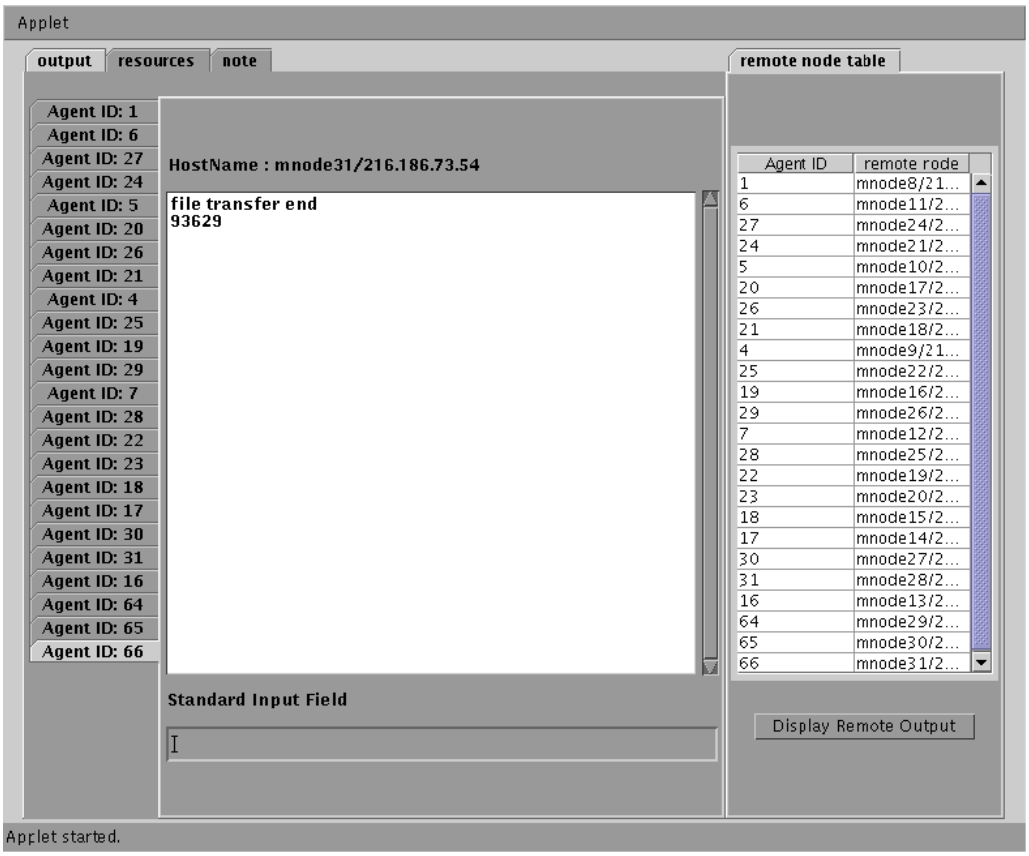

図 4.3: 各ユーザプログラムからの標準出力画面

4.3 機能に関する考察

ここでは、本研究で実装したユーザインタフェースの実装前と実装後での利便性の違いについて述べる。

AgentTeamwork インタフェースが実装される以前では、ユーザは、長いシェルコマンド、あるいはそれに相当するシェルスクリプトファイルを、ジョブを投入するたびに作成する必要があった。本インタフェースを実装することで、これらのユーザによる負担を軽減することができた。また、AgentTeamwork は、ユーザに意識させることなく、ジョブの投入から実行結果を得るまでの流れを提供するべきである。本インタフェースは、コマンダエージェントとの通信を行うことで、ユーザと、遠隔計算機上のユーザプログラムとの通信を実現した。これにより、ユーザはインタフェース下位層の、エージェント同士の通信を

意識する必要がなく、インタフェース上での操作だけですべての処理をすることができるため、その処理の流れを隠蔽することができた。

　また、入出力機能の開発により、ユーザとユーザプログラム間の標準入出力の交換を、インタフェース上の個別の画面で操作することができるため、ユーザは、対応するユーザプログラムごとに、標準入出力用のコンソールなどを用意する必要が無く、利便性が向上されているといえる。

第 5 章　関連研究

本章では、AgentTeamwork の機能を検証する。その比較対象として、入出力では、代用的なグリッド計算システムである、Condor[5][6]、Legion[7][8]について、ユーザインタフェースでは、グリッドポータルである GridSphere[9][10]、uPortal[11]について述べる。

5.1　入出力機能

本節では、Condor と Legion の概要について述べ、それらの入出力機能について述べる。

1. Condor

   Condor とは、ユーザの逐次ジョブを実行するためのジョブスケジューリングシステムである。ユーザは、Condor を使ってジョブを実行する際に、いくつかの実行環境の中から、適切な環境を選んでジョブを実行することができる。ここでは、標準的に利用される、Vanilla 実行環境、および Standard 実行環境の入出力に注目する。

   Vanilla 実行環境は、通常の逐次ジョブを実行する際に用いられ、Standard 実行環境は、チェックポイント･リスタートジョブを実行する際に用いられる。チェックポイント･リスタートジョブとは、ユーザのジョブが、他のジョブによって割り込みを受けた場合に、自動的にジョブのチェックポイントをとることによって、チェックポイントがとられた時点から再開することができるジョブのことである。

   Vanilla 実行環境は、ユーザが選択可能な 2 種類の入出力機能を提供する。一つは、共有ファイルシステムを使用する方法である。この方法を使用する場合、使用するファイルが NFS や AFS などのファイル共有システムによって共有されていなければならない。もう一方は、ジョブが実行される前に、すべてのファイルをジョブが実行されるノードに転送する方法である。この転送は、リモート入出力ソケットを用いる。この方法は、ジョブが実行されるノードからローカルノードに向かってソケット通信を行い、ローカルノードのファイルにオンラインアクセスを行うもの

である。

一方、Standard 実行環境は、遠隔ノードで実行されるジョブの入出力に対して、Condor が提供するリモートシステムコールを使用する。このリモートシステムコールは、遠隔ノードで実行しているユーザジョブが発行するすべてのシステムコールを、ジョブが投入されたローカルノードに転送する機能を持つ。つまり、ユーザジョブが遠隔ノード上で、実行時に発行したシステムコールは、ローカルノードへ転送され、そこで実行された後、その結果だけが再度遠隔ノードへ返送される。結果として、Standard 実行環境下における Condor の入出力は、プログラム中で、ファイルアクセス、および標準入出力を必要とするときにのみ、随時、遠隔からローカルノードへデータの送受信を行う。

2. Legion

   Legion は、このシステムを導入している計算機上の資源にアクセスできる、オブジェクト指向のオペレーティングシステムである。Legion を使用することで、ユーザは、自分が今アクセスしている資源がどこに存在しているかを全く意識することなく、ファイルシステムや資源管理などの OS サービスを受けることができる。Legion で定義されるオブジェクトとは、計算機資源そのものを表す。このオブジェクトを配置する空間は、context space と呼ばれ、Legion システムを導入している任意の計算機上に配置される。Legion における入出力機能は、Legion NFS Daemon（lnfsd）とよばれる独自のファイルシステムによって実現されており、このシステムはその context space を、利用するユーザに提供する。これにより、ユーザは、NFS や AFS と同じように、その計算機に存在するファイルにアクセスすることができる。すなわち、ユーザのプログラムが、Legion システムを使って遠隔地で実行された場合、その入出力は、通常の NFS を利用するのと同様に利用することができる。

Condor および Legion は、両者ともネットワークを介したファイル転送の自動化をはかっている。しかし、Condor は、NFS や AFS などの共有ファイルシステムが無い場合、必ずその入出力は、ローカルノードと遠隔ノードでの 1 対 1 通信となるのに対し、Legion は、独自にファイルシステムを構築していることにより、すべての入出力が、ファイル共有という形で行われる。

5.2 ユーザインタフェース

グリッドポータルとは、専門的な知識を必要とすることなく、ブラウザから、メニューなどを用いたグラフィカルユーザインタフェースを使用することにより、ネットワーク上の資源を簡単に利用することができるシステムである。本節では、グリッドポータルの例として、GridSphere、uPortal の概要について述べる。

1. GridSphere

   GridSphere は、ウェブサーバの構築に必要なフレームワークを、サーブレットを使って提供するシステムである。図 5.1 に示すように、このフレームワークは、（1）各種のウェブアプリケーションを実行するポートレット・コンテナ、（2）ユーザのログイン情報を管理するコアサービス、および、（3）ウェブアプリケーションの表示を管理するレイアウトマネージャの 3 つの実行環境を管理する。このうち、GridSphere ポートレット・コンテナは、グリッド計算に不可欠なリソース管理、ジョブ投入、ファイル参照の各サービスを、ポートレットと呼ばれるオブジェクトとして実行するための環境である。ポートレットは、ユーザが独自に開発することもできるが、基本的なグリッド計算機用のサービスは、グリッドポートレットとしてシステムからすでに提供されている。ここでは、このグリッドポートレットの機能について述べる。

   GridSphere が提供するグリッドポートレットには、以下の 3 つがある。

   - リソースブラウザポートレット

     GridSphere を通して利用可能な資源を調べ、どの様なサービス、およびソフトウェアがそれらの資源で利用可能であるかを見ることができる。

   - ジョブ投入ポートレット

     GridSphere を通して投入されたジョブを監視し、その履歴、および完了通知を受ける。

   - ファイルブラウザポートレット

GridSphere を通して、遠隔ノードのファイルシステム上にあるファイルのリストアップ、改名、およびディレクトリの作成や、ファイルのアップロード、ダウンロードを可能にする。

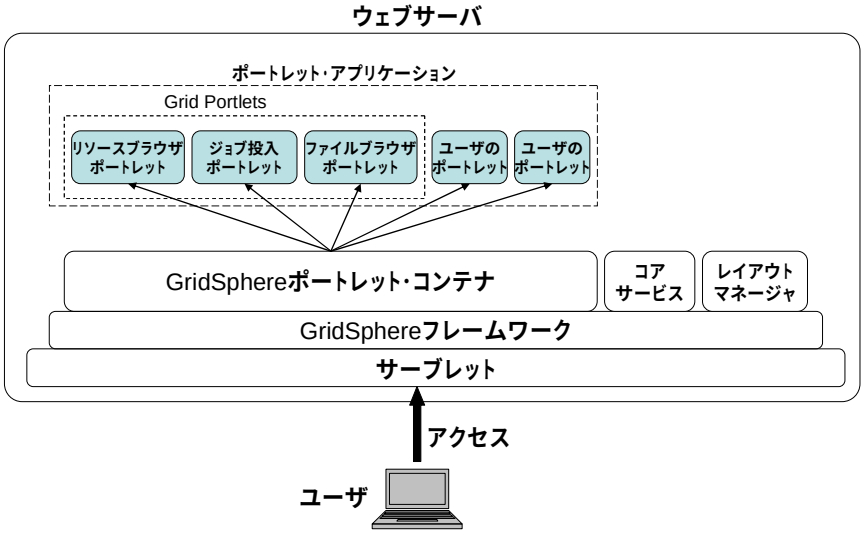

図 5.1:　GridSphere グリッドポータルの構成の例

2. uPortal

uPortal は、図 5.2 に示されるように、サーブレット上で動作するポータルアプリケーションを構築するためのフレームワークである。uPortal では、ユーザに提供する情報を、チャネルと呼ばれる単位で管理する。このチャネルには、その処理に応じて 7 種類のものが提供されている。これらのチャネルのいくつかは、既存のアプリケーション（ウェブアプリケーション、XML、またはアプレットなど）をそのままチャネルとして利用することが可能なため、汎用性が高い。このほかに、uPortal では、グリッド計算機用のチャネルとして、遠隔チャネルプロシキと呼ばれるチャネルを提供している。これは、本来自分が行うべき処理を、遠隔のポータル上にあるチャネルに任せる。これを使用することで、ユーザは遠隔ノード上の計算機資源を使用して、処理を行うことができ、処理結果のみを uPortal のチャネル上に表示することができる。

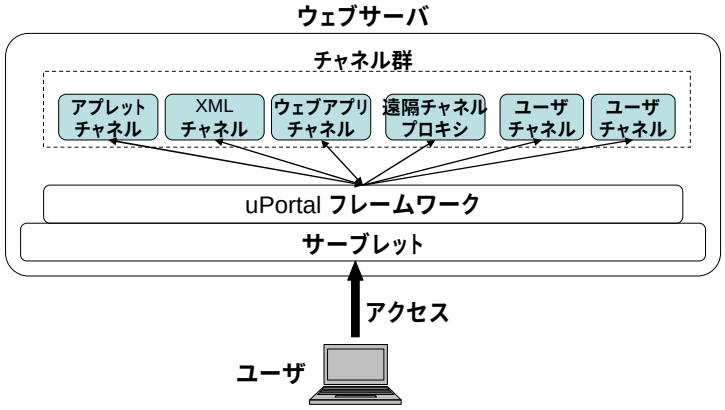

図 5.2: uPortal の構成の例

GridSphere および uPortal は、両者とも、ポータルを利用することによって、ネットワーク上に広がる様々な資源を、グラフィカルな操作のみで簡単に利用することができる。GridSphere は、その情報提供の単位として、ポートレットのみをサポートしているのに対し、uPortal は、既存のアプリケーションを利用できるという点から、より汎用性のあるポータルを構築することが可能であり、それにともなって多くのサービスを受けることができる。

5.3 AgentTeamwork との比較

本節では、はじめに、Condor、および Legion と AgentTeamwork の入出力機能を比較し、AgentTeamwork における入出力機能の優位性について述べる。次に、GridSphere、および uPortal と AgentTeamwork のユーザインタフェースについて比較し、本研究で開発したインタフェースが、AgentTeamwork の、サーバ非依存という特徴に適合していることを述べる。

1. 入出力機能

   Condor の入出力機能は、NFS を利用する方法と、直接ファイルを転送する方法の 2 つがあるが、ユーザのジョブを実行する環境に NFS が存在しない場合、すべての入

出力は、遠隔ノードとの 1 対 1 通信となる。よって、同時に複数のジョブを投入した場合、遠隔ノード上のすべてのジョブからローカルノードに対してファイル転送要求が送られるため、ローカルノードの通信負荷は増大する。Legion における入出力機能は、ファイル共有で行われるため、Condor に比べるとその負荷は小さいと言える。しかし、Legion の場合は、実行するジョブ数の増大にともなって、複数のジョブで同じファイルを共有するため、ファイルサーバの負担が大きくなる。

一方、AgentTeamwork のファイル転送は、エージェントの木を使用して行われる。このため、ジョブ数、およびエージェント数の増加にともなう、一つのノードあたりにかかる通信負荷は、最大でも、自分の子供の数に限定される。Condor とファイル転送の面で比較すると、ファイルサイズ、およびジョブ数が増加するたびに、この差が顕著に現れる。また、ファイルサーバの負担について注目した場合、AgentTeamwork は木を使ってファイルを転送するため、転送するファイルの読み込みを行うのは、木の根となるコマンダエージェントのみである。したがって、AgentTeamwork では、ファイルの読み込み回数は、わずか 1 回だけであり、コマンダエージェントの子孫となるエージェントは、そのファイルを同じファイルサーバから読み込む必要が無い。このため、ファイルサーバにかかる負荷は、Legion に比べて極めて小さい。以上により、大容量のデータファイルを頻繁に扱うグリッド計算システムにおいて、AgentTeamwork の入出力機能は、Condor、Legion に比べて、ファイル転送、およびファイル読み込み回数の面で、効率的であると予測できる。

2. ユーザインタフェース

GridSphere および uPortal は、両者ともサーブレット上で使用されるユーザインタフェースである。このため、ユーザは、インターネットを観覧する手段さえあれば、誰でも特別な技術を必要とすることなくネットワーク上の様々な資源を利用することができる。しかし、AgentTeamwork は、サーバを介さないシステム設計となっているため、これらのポータルを使用する場合、ユーザの各端末にウェブサーバを導入する必要が生じる。この作業は、ユーザにとって大きな負担となり、AgentTeamwork を使用する際には自分の端末上で常にサーバを起動しておく必要がある。この点を考慮すると、本研究で開発した AgentTeamwork のインタフェー

スは、アプレットを用いて開発されているため、各端末にウェブサーバを導入する必要も無く、ウェブからのダウンロードのみで使用できるため、ユーザにとってインタフェースの設置に関する負担が少ない。よって、サーバを介さないという観点から見て、本研究で開発したユーザインタフェースは、AgentTeamwork に適しているといえる。

第 6 章　結論

本研究では、従来の AgentTeamwork が備えていなかった、入出力機能の開発、および、ユーザインタフェースの開発を行った。入出力機能の開発では、エージェントの木を利用したファイル転送を行うことにより、ファイルサーバの負荷分散、および、ファイル転送の並列化を実現し、NFS のファイル転送と比較して、256M バイトのファイルにおいて約 7 倍の速度で転送できることを示した。また、GridFile を提案し、遠隔ノード上での入出力データの排他制御を実現した。ユーザインタフェースの開発では、サーバを介さない AgentTeamwork の特徴を活かした、Java アプレットでのユーザインタフェースを開発することにより、ユーザジョブの投入を容易にした。また、各ユーザプログラムから生成される標準出力データの表示、および、各ユーザプログラムへ届けられる標準入力データの転送要求を、インタフェース上の個別の画面から操作できるようにした。

しかし、AgentTeamwork の入出力、およびインタフェースには、現在 3 つの問題点が残されている。一つ目は、すべてのエージェントに入力ファイルが転送されるという点である。このため、ユーザプログラムごとに異なるファイルを扱いたい場合、不必要なファイルまで転送される。2 つ目は、ユーザ自身がファイル分割数を決定できないという点である。このため、ファイルサイズの動的変化に対応することができない。そして 3 つ目は、エージェントが移動した際に、ディレクトリに格納された入出力ファイルは移動しないという点である。エージェントは、移動先の計算機障害により別の計算機へ移動することがある。この場合、メモリに格納されているファイルは、スナップショットとしてブックキーパエージェントが保持しているため、そのファイルデータも一緒に移動するが、ディレクトリに格納されている場合は、スナップショットがとられないため、一緒に移動させることができない。これらの問題点を解決するため、（1）各ユーザプログラムへ転送するファイルの自動判別機能の追加、（2）動的なファイル分割数の決定、（3）ディレクトリに格納された入出力ファイルもスナップショットを取る、の以上 3 点が、今後の課題となる。

謝辞

はじめに、AgentTeamwork は、米国立科学財団(National Science Foundation) Shared Cyberinfrastructure プログラムの支援（登録番号 0438193）を受けて、ワシントン大学で遂行されているプロジェクトであり、本卒業研究は、AgentTeamwork プロジェクトの一環として、ワシントン大学バセル校および愛媛大学との間の学術協定にもとづく交換留学制度のもとで、行われたことを記します。

本研究の全過程を通じ、懇切丁寧な御指導を賜わった 小林真也教授に心から感謝致します。本論文の作成中、様々な御助言を頂いた、柏木紘一助手に心から感謝申し上げます。

留学中、本研究の直接的な御指導、および適切な御助言を頂き、貴重な御時間を割いて頂いた、ワシントン大学バセル校 福田宗弘助教授に厚く御礼申し上げます。

本研究を進めるにあたり、留学先での生活に多大な協力をして頂いた、ホストファミリーの方々に感謝致します。

本論文の作成に、有益な知識となる授業を手配して頂いた、ワシントン大学バセル校計算ソフトウェアシステム学科 ジャクル学科長、メスキ女史、ハンタ女史に、心より感謝致します。

ここに記して、以上の皆様に感謝の意を表します。

参考文献

[1] Munehiro Fukuda, Koichi Kashiwagi, Shinya Kobayashi, "AgentTeamwork: Coordinating Grid-Computing Jobs with Mobile Agents", In Special Issue on Agent-Based Grid Computing, International Journal of Applied Intelligence, to appear in 2006.

[2] Munehiro Fukuda, Koichi Kashiwagi, Shinya Kobayashi, "The Design Concept and Initial Implementation of AgentTeamwork Grid Computing Middleware", In *Proc. of IEEE Pacific Rim Conference on Communications Computers and Signal Processing - PACRIM'05*, pp. 255-258, 2005

[3] Ian Foster and Carl Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, pp. 15-21, Morgan Kaufmann Publishers, 1999.

[4] H.M.Deitel and P.J.Deitel, *JAVA HOW TO PROGRAM*, Prentice-Hall, 1997.

[5] Condor Project Homepage, http://www.cs.wisc.edu/condor/, 2006.

[6] Condor Version 6.6.10 Manual, http://www.cs.wisc.edu/condor/manual/v6.6/, 2006.

[7] Brian S. White, Andrew S. Grimshaw and Anh Nguyen-Tuong, "Grid-Based File Access: The Legion I/O Model", In *Proc. of the Ninth IEEE International Symposium on High Performance Distributing and Networking (NPCN 2001)*, pp. 32-41, June 2001.

[8] Brian S. White, Marty Humphrey and Andrew S. Grimshaw, "An Interposition Agent for the Legion File System", Department of Computer Science, University of Virginia, http://www.csl.cornell.edu/~bwhite/papers/masters.pdf, January, 2002.

[9] Novotny, Jason, Russell, Michael and Wehrens, Oliver, "GridSphere: a portal framework for building collaborations", Concurr. Comput.-Pract. Exp., Vol. 16, pp. 503-513, 2004.

[10] Jason Novotny, Michael Russell and Oliver Wehrens, “GridSphere: An Advanced Portal Framework€,35 http://www.gridsphere.org/gridsphere/wp-4/Documents/France/gridsphere.pdf, 2006.

[11] uPortal home page, http://www.uportal.org/, 2005.

[12] mpiJava home page, http://www.hpjava.org/mpijava.html, 2004.

[13] Milojicic, D. et al., *Processes, Computers, and Agents,* Addison-Wesley, 1999.