社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2009－ARC－182 (11)
2009－HPC－119 (11)
2009／2／27

ネットワーク帯域予約とOS仮想化機構を用いた分散アプリケーション実行環境に向けて

中田秀基† 高野了成†
竹房あつ子† 工藤知宏†

複数のサイトにまたがって計算資源，ネットワーク資源を動的に確保し，アプリケーションを実行することは以下の理由により困難である．1) ネットワークの非対称性、2) サイト間の非均質性，3) 必要情報の実行時確定，4) アプリケーションのマルチホームネットワーク非対応．このため，現在のグリッド環境では既存のアプリケーションをそのまま実行することは難しい．我々は，予約ベースで計算資源とネットワーク資源を確保し，その上にごく一般的なクラスタ内の環境と類似した環境を構築するグリッドミドルウェアの構築を行っている．このグリッドミドルウェアを用いると，既存のアプリケーションを修正すること無く実行することが可能となる．本稿ではこのグリッドミドルウェアのアプリケーション実行フレームワークの設計とプロトタイプ実装について述べる．プロトタイプを用いた実行の結果，フレームワークの設計の妥当性を確認した．

Towards Distributed Application Execution Environment with Reserved Network Bandwidth and OS Virtualization

Hidemoto Nakada ,† Ryousei Takano ,† Atsuko Takefusa †
and Tomohiro Kudoh †

To execute applications over dynamically allocated network resources and computing resources from several administration sites are difficult by several reasons include: 1) asymmetric network reachability, 2) sites administration heterogeneity, 3) dynamic determination of required connection information, 4) multi-home network unaware applications. Therefore, it is not possible to execute existing non-grid-aware applications on such an environment without any modification. We have been developing a grid middleware that provide an environment similar to ordinal cluster environment on top of computing and network resources co-allocated by reservation basis. On the environment applications for cluster environment will just run without any modification. This paper describes the middleware focusing on its application execution framework, and prototype implementation. The prototype showed that the architecture of the framework is promising.

1. はじめに

グリッド技術の目的の一つとして，複数のサイトにまたがって計算資源，ネットワーク資源を確保し，それらにまたがった一つのアプリケーションを実行することが挙げられる．これは下に述べるいくつかの理由によってそれほど簡単ではない．

(1) **ネットワークの非対称性**
NATなどによってネットワークの自由な相互接続性が確保できない．

(2) **サイト間の非均質性**
各サイトは別々の管理主体によって管理されているため環境が非均質である．たとえば，同一のユーザであってもユーザ名がサイトごとに違うことがありうる．

(3) **必須情報の実行時確定**
ノード間の連携に必要とされる情報が事前に確定せず，実行時になって初めて確定する．

(4) **アプリケーションの不備**
一部のアプリケーションは，単一のネットワークインターフェイスを前提としており，複数のネットワークインターフェイスを持つ環境に適応できない．

このため，現在のグリッド環境では，アプリケーションを専用のミドルウェアを用いて記述する必要があり，既存のアプリケーションをそのまま実行することができない．

† 産業技術総合研究所 National Institute of Advanced Industrial Science and Technology (AIST)

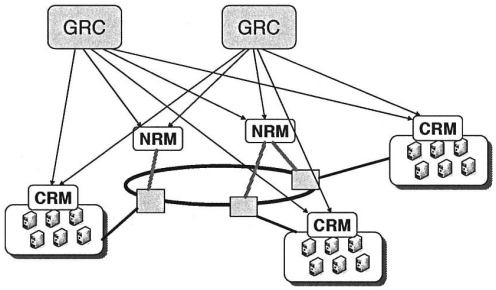

図 1 GridARS の概要

我々は，この問題を解決するために，アプリケーションの実行環境を整えるグリッドミドルウェアの開発を行っている．このミドルウェアは予約ベースで計算資源とネットワーク資源を確保し，確保した資源上に，ごく一般的なクラスタ内の環境と類似した環境を構築する．これによって，既存のアプリケーションを修正すること無く実行することが可能となる．

本稿ではこのミドルウェアのアプリケーション実行フレームワークの設計に関して詳述する．また，このフレームワークのプロトタイプ実装について述べる．次節以降の構成は次の通りである．2 節で研究の背景について述べる．3 節でフレームワークの設計について述べる．4 節でプロトタイプ実装について述べる．5 節で関連研究に言及し，6 節にまとめを示す．

2. 背　　景

われわれは，ネットワーク資源と計算機資源を事前予約を用いて同時に確保し，アプリケーションを実行するミドルウェア GridARS を開発しており，日米数サイトを国際回線で接続した実証実験を行ってきた [1)2)3)]．

GridARS の概要を図 1 に示す．各計算機ノードは CRM(Computing Resource Manager) と呼ばれるモジュールで管理される．CRM は，バックエンドとして PluS[4)] や PBS Professional などの事前予約可能なバッチキューイングシステムを用いてノードを事前予約する．ネットワークは，NRM(Network Resource Manager) と呼ばれるモジュールで管理される．NRM は内部に予約テーブルを持ち，ネットワーク装置を制御して帯域の確保されたネットワークを各ユーザに割り当て，参加ノードの VLAN を制御してユーザプロセスに提供する [5)]．GRC(Global Resource Coordinator) はこれらのモジュールと連携して資源の同時確保を行う．複数の GRC が NRM/CRM を共有することも可能である．さらに，GRC が階層的に構成されることもあり得る．

前節でも述べたように，複数のサイトからなる環境上でアプリケーションを実行する際には幾つかの困難がある．このうち (1) に関しては，ネットワーク資源を動的に確保する我々のミドルウェアにおいては問題にならないが，残る (2)-(4) の解決が課題となっていた．以下それぞれに関して詳述する．

2.1 サイト間の非均質性

各サイトは独立した管理組織に属するクラスタである．このため前述したユーザ名の問題以外にも様々な問題がある．とくに問題となりうるのが相互ログインの設定である．仮にネットワーク的に到達可能であっても，事前に相互に ssh の公開鍵を交換しておかなければ相互に ssh でプロセスを起動することはできない．サイト数がある程度の数になると，この作業は非常に煩雑となる．

2.2 必須情報の実行時確定

一般に並列アプリケーションを実行する際には，すべての参加ノードが他のすべての参加ノードのアドレスを知る必要がある．例えば通常のバッチキューイングシステムで管理されたクラスタ環境では，バッチキューイングシステムがノードのリストを作成し，ファイルとしてアプリケーションに引き渡す．

しかし，複数サイトを用いたシステムではこれを実現することは容易ではない．情報が複数サイトに分散しており収集，分配をおこなわなければならないだけでなく，各サイト内でのジョブの実行は独立したバッチキューイングシステムが管理しており，ミドルウェアが直接管理することができないからである．

前述の実験では，Ninf-G[6)]，GridMPI[7)] といったグリッドミドルウェアを用いた．GridMPI では，起動時に IMPI と呼ばれるプロトコルを利用することでノード情報のグローバルな共有を実現している．このプロトコルでは，IMPI サーバと呼ぶすべてのサイトから到達可能で既知のノードで動作するデーモンを利用する．各サイトの代表ノードがこのデーモンを経由して情報を交換することで，各ノードが全ノードの IP アドレスを知ることができる．GridRPC システム Ninf-G では，通信はクライアントを中心としたツリー構造となる．このため，各ノードがクライアントのアドレスを知っていれば十分で，各ノードが IP アドレスを共有する必要はない．このようにグリッド環境向けの特殊なミドルウェアを用いればこの問題を回避することができるが，一般的なクラスタを対象としたアプリケーションを，そのままグリッド環境で実行することは難しい．

2.3 アプリケーションの不備

あるノード A 上のプロセス P が他のノード B 上のプロセス Q からの接続を受けたい場合を考える．この場合プロセス P は，オープンしたポート番号とともに，ノード A のホスト名もしくは IP アドレスをプロセス Q に提示する必要がある．

このとき，ノード A が複数のネットワークインターフェイスを持っているとすると，提示するべきホスト

名もしくはIPアドレスが一意に定まらなくなってしまう．これを決定する方法は一般にはなく，何らかのヒント情報を外部から得る必要がある．しかし多くのアプリケーションはそのようなヒント情報を取得するインターフェイスを持たず，結果としてマルチホーム環境では動作しなかったり，動作しても不適切なネットワークを用いて低速に動作する可能性がある．

3. 設　　計

前述の問題を解決するために，開発中のミドルウェアにアプリケーションの実行を補助するフレームワークを組み込むべく設計を行った．目的は，通常のクラスタ上で稼働するアプリケーションをユーザに負荷をかけることなく実行できるグリッドミドルウェアを実現することである．

並列アプリケーションとしては，ホストファイルを読み込み，sshでプロセスを起動するアプリケーションを前提とする．また，マルチホームネットワークには対応していなくても良いものとする．

3.1 要　　請

アプリケーション実行フレームワークが提供するべき環境への要請を整理する．上述の並列アプリケーションを実行するためには，1) すべてのノードのIPアドレスのリストが互いに配備されており，2) すべてのノードのsshのユーザ公開鍵，ホスト公開鍵が交換されて配備されている，必要がある．

さらに，3) ノードが使用するIPアドレスはすべてのサイトで重複があってはならない．このためには，サイト間での，使用するアドレスレンジに関するネゴシエーションが必要となる．

また，マルチホーム問題に対応するためには，4) アプリケーションプロセスからは単一のネットワークインターフェイスのみが見えるように制御する必要がある．

3.2 設　　計

上述の要請を実現するフレームワークの概要を図2示す．本フレームワークでは，各サイトのキューイングシステムでは直接ジョブを実行せず，いわゆるPilotジョブとしてアプリケーションの実行環境を整えるデーモンであるNodeManager(図中NM)を実行する．NodeManagerはCRMに対してネットワーク接続を行い，CRMの制御下に入る．また，仮想化機構を用いてアプリケーション実行環境を整備し，実行環境上でユーザのジョブを実行する．

3.2.1 IPアドレスの決定と収集分配

要請 (3) で述べたように，各ノードが使用するIPアドレスは，全サイトで重複しないよう管理されなければならない．しかし，各サイトで利用可能なアドレスは各サイトが独立して管理する必要がある．これらを両立させるため，下記の手続きで各ノードのIPアドレスを決定する．

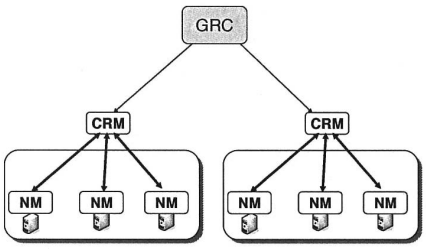

図2 アプリケーション実行フレームワークの概要

- 予約手続き開始前
  各サイトのCRMはそのサイトで利用可能なIPアドレスのレンジを，サイト情報の一部として公開する．
- 予約時
  GRCは資源予約時に利用したいIPアドレスのレンジを各CRMに対して指定する．このさいGRCが各サイトのアドレスが重複しないことを保証する．予約時点で，そのアドレスレンジが利用できなかった場合には，予約手続きは失敗する．
- 実行時
  CRMは指定されたIPアドレスプールのなかから，各ノード上のNodeManagerにIPアドレスを割り当てる．
  さらに要請 (1) のIPアドレスの共有を実現するために，実際に利用されたアドレスの収集と分配を3者が連携して行う．GRCは，予約時間になるとCRMに対してポーリングを行い，実際に割り当てたIPアドレス集合を取得する．すべてのサイトのCRMから情報が集まった時点で，アドレスのリストを作成しCRMに与える．CRMはそれを各NodeManagerに提供する．NodeManagerはこの情報を仮想環境内にセットし，アプリケーションを実行する．

3.3 ssh関連情報の生成と共有

要請 (2) を満たすためには，仮想環境内で利用されるsshd/sshのホスト公開鍵，ユーザ公開鍵を，実行時に生成し，共有する必要がある．

NodeManagerは仮想化環境内にホスト用の暗号鍵ペア，ユーザ暗号鍵ペアを作成する．CRMは生成された公開鍵を収集し，前項で述べたIPアドレスと同じ手順ですべてのノードで共有する．共有されたホスト公開鍵をknown_hostsファイルに，ユーザ公開鍵をauthorized_keysに登録することで，すべてのノードからすべてのノードにsshでのログイン，およびプロセス起動が可能になる．

3.4 マルチホームネットワークの隠蔽

要請 (4) を満たすためには，複数存在するネットワークインターフェイスを隠蔽する必要がある．このために仮想化環境を用いる．仮想化環境内に選択的に

単一のネットワークを提供することで，任意のマルチホーム非対応アプリケーションを実行することが可能になる．

この際に仮想化環境とホスト計算機環境のディレクトリ構成は基本的に共有することでユーザによるアプリケーションの事前配備を容易にする．

3.5 アプリケーション実行までの流れ

以下にユーザのアプリケーションが実行されるまでの流れをまとめる．

(1) ユーザは事前にアプリケーションを各サイト上に配備する．
(2) 各サイトの CRM は利用可能な IP アドレスをアドバタイズする．
(3) ユーザは GRC に対して計算資源，ネットワーク資源を予約し，アプリケーションの実行を依頼する．
(4) GRC は各サイトで利用する IP アドレスレンジを決定し，そのレンジを指定して CRM に対して予約を行う．CRM は，バックエンドのキューイングシステムに対して予約を行い，NodeManager をジョブとして投入する．
(5) 実行時間になると，NodeManager が各ノード上で起動し，CRM にコールバックする．NodeManager は仮想化環境を整え，ssh の暗号鍵ペアを用意する．
(6) 前述のように，GRC が主導して，IP アドレス情報，ssh 公開鍵が共有されセットされる．
(7) すべての準備が整った段階で，GRC が主導して各ノード上でアプリケーションのジョブプロセスを起動する．ただし多くの場合は，起動の起点となるノードだけでジョブを起動すれば十分である．

4. 実　　装

4.1 仮想化層の選択

仮想化機構としては，大別して計算機仮想化機構と OS 仮想化機構が存在する．一般に前者と比較して後者のオーバヘッドは著しく小さい．このため我々は OS 仮想化機構を利用することとした．

オープンソースで利用可能な OS 仮想化層としては現在 Linux-VServer[8] と OpenVZ[9] が存在する．この二者のうち，どちらを利用するべきかを決定するために，両者のネットワーク性能を測定した．

測定では 1GbE として Broadcom BCM5722 を，10GbE として Myricom Myri-10G を用いた．MTU はそれぞれ 1500 バイト，9000 バイトとした．いずれの場合も 2 台の計算機をネットワークケーブルで直結しており，間にスイッチなどを介していない．使用した計算機は送受信双方とも，CPU に Intel quad-core Xeon E5430/2.66GHz x 2 をメモリを 4GB (DDR2-667) 搭載したものを用いた．OS としてはいずれの場合も，ホスト，ゲストともに，ubuntu 8.10 server を用いている．使用した仮想化機構のバージョンを表 1 に示す．vanilla は仮想化機構を用いない場合である．

表 1 仮想化機構のバージョン

機構	バージョン
vanilla	2.6.27-9-server
Vserver	2.6.27.10-vs2.3.0.36.2
OpenVZ	2.6.27.10-openvz (git)
Xen	Xen 3.3, Dom0 2.6.26.1

表 2 仮想化機構の通信性能-1G [TCP/UDP]

	Mbps	SCPU	RCPU
vanilla	941.38 / 957.2	0.76 / 1.99	0.33 / -0.25
VServer	941.37 / 957.2	1.00 / 1.77	0.16 / -0.19
OpenVZ	941.07 / 957.2	1.90 / 3.02	1.57 / 2.42
Xen	949.19 / 957.2	5.99 / 0.25	4.54 / 3.50

表 3 仮想化機構の通信性能-10G [TCP/UDP]

	Mbps	SCPU	RCPU
vanilla	9525.94 / 8051.2	8.13 / 8.13	7.84 /7.84
VServer	9521.79 / 8149.9	9.11 / 14.11	7.51 / 7.11
OpenVZ	2049.89 / 3005.0	10.12 / 12.60	2.74 / 3.55
Xen	1011.47 / 6252.5	0.85 / 99.98	1.38 / 8.04

測定したネットワーク性能および CPU 消費率を表 2, 表 3 に示す．測定には Netperf[10] を用い，3 回実行し，最良値を提示している．図中の SCPU/RCPU はそれぞれ送信側計算機および受信側計算機の CPU 使用率である．各項の数字は左が TCP を用いた場合の値，右が UDP となっている．一部のデータで CPU 負荷値が小さい負の値になっているが，これは負荷が 0 に非常に近いと考えられる．

OpenVZ に関しては仮想環境内で CPU 使用率を正確に取得することができないため，仮想環境で NetPerf を実行した際のホスト環境の CPU 使用率を示した．

参考のため，2 つの OS 仮想化機構に加えて，代表的な計算機仮想化機構の一つである Xen[11] での結果も併記する．Xen の結果は仮想計算機内で測定されているためクロックの精度に問題がある可能性がある．また，CPU 利用率は DomU が使用可能な 1 コアのみに対するものである点に注意されたい．

1GbE の場合にはいずれの機構を用いても，ほぼネットワークの性能をフルに引き出すことができている．

10GbE では VServer は vanilla と同様の性能を引き出すことができている．これに対して OpenVZ は TCP で 1361Mbps と低い性能にとどまっている．このためわれわれは，VServer を用いることとした．

4.2 仮想化環境内ファイルシステムの構成

仮想化環境とホスト環境は，ある程度環境を共有しながら，隔離されている必要がある．アプリケーショ

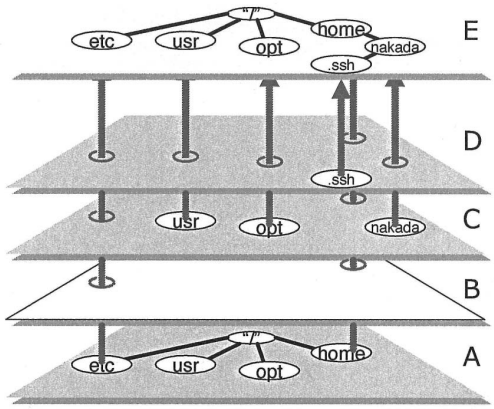

図 3 ファイルシステムの構成

ンの実行に必要な/usr, /opt, ユーザのホームディレクトリを共有しつつ，ユーザ名を変更し，ユーザの .ssh ディレクトリの書き換えを可能にしなければならない．

われわれは，これを実現するために，スタッカブルファイルシステムの一つである Aufs(Another UnionFS)[12] と，バインドマウントを用いて仮想化環境のファイルシステムを構成した．Aufs を用いることでテンプレートとなるディレクトリ構造を対象環境のディレクトリ構造として読み出しのみで公開しつつ，その上に読み書き可能なファイルシステムを構成することができる．バインドマウントを用いることで，公開されているディレクトリ構造の一部に別のディレクトリ構造を重ねて隠蔽することが可能となる．

図 3 に構成するファイルシステムの様子を示す．テンプレート (図 3A) ととなるディレクトリツリーは，VServer が提供するコマンドで CentOS のイメージを作成している．これには数分ないし数十分の時間がかかるが事前に一度行っておくだけでよい．この上に書き込みを受けるための空のディレクトリツリー (図 3B) を重ねている．たとえば，/etc に書き込みを行うと，実際にはこのツリーに書き込みが行われる．その上に，バインドマウントを用いて，/usr, /opt, ユーザディレクトリを重ねる (図 3C)．さらに，ユーザの.ssh ディレクトリは別のディレクトリを重ねている (図 3D)．これは，ssh の環境を整備する際に，仮想環境内だけで利用する.ssh 関連の鍵，設定ファイルを設置するためである．これらをが重なり合うことで仮想化環境内のディレクトリツリー (図 3E) が構成される．

4.3 アプリケーション実行フレームワークプロトタイプの実装

前節で述べたアプリケーション実行フレームワークの設計の妥当性を確認するために，実行フレームワークに関する機構を抽出し，プロトタイプを作成した．作成したモジュールは GRC, CRM, NodeManager の 3 つである．

表 5 NodeManager の状態

コマンド名	意味
INITIAL	開始直後
INITIATING	INIT コマンドで遷移．ディレクトリツリー構成，鍵生成中
HOST_READY	公開鍵の生成が完了した状態．GET_KEYS を待っている
KEYS_READY	SET_KEYS で遷移．仮想環境の起動可能状態を示す
STARTING_UP	仮想環境起動中
RUNNING	仮想環境起動完了
SHUTTING_DOWN	仮想環境終了中
DONE	仮想環境停止完了
FAILED	何らかのエラーで状態遷移に失敗

GRC と CRM は Java を用いて，NodeManager は Python で実装した．GRC と CRM は Java RMI を用いて同期通信を行う．CRM と NodeManager はテキストベースの非同期ノーティフィケーションをゆるすプロトコルを用いて通信する．構造データのエンコーディングには JSON[13] を用いている．

CRM,NodeManager 間の通信コマンドを表 4 に，NodeManager の持つ状態とその意味を表 5 に示す．

実装したプロトタイプを用いて，簡単な実験を行い動作を確認した．実験環境は単一のノードで，GRC, 2 つの CRM，4 つの NodeManager を実行し，簡単な MPI のアプリケーションを実行した．この結果，本フレームワークの設計の妥当性が確認できた．

5. 関連研究

仮想化技術を用いて，ネットワーク的に独立した環境を構築する試みとしては，PlanetLab[14] や，尾崎らによる研究 [15] がある．前者は本研究と同様に Linux-VServer を，後者では KVM を仮想化レイヤとして使用している．

これらの研究と本研究の相違は目的と提供される環境のライフタイムとホスト環境からの隔離性にある．これらの研究では，環境のライフタイムは長く，ホスト環境からは完全に隔離されていることが重要である．これに対して，本研究が提示する枠組みは，単一アプリケーションの実行時間内に高速に環境を立ち上げ，停止することに一つの眼目を置いている．またホスト環境とは適度に隔離しつつ，ディレクトリツリーの大半を共有して事前にアプリケーションをインストールすることを可能としている．

6. おわりに

我々が実装中の，予約ベースで確保した計算資源とネットワーク資源上に，ごく一般的なクラスタ内の環境と類似した環境を構築するグリッドミドルウェアに関して，特にアプリケーション実行フレームワークの設計に焦点をあてて述べた．また，アプリケーション

表 4 NodeManager のコマンド

コマンド名	入力	出力	意味
INIT	IP アドレス, VLAN タグ, 使用ユーザ名	N/A	仮想環境で用いる値を与える．NodeManager は ack を返した後仮想環境のセットアップを行い，ユーザ鍵ペア，ホスト鍵ペアを生成し，終了後ステイトを HOST_READY に変更する．ステイトの変更は非同期のノーティフィケーションとして CRM に送られる．
GET_KEYS	N/A	ユーザ公開鍵，ホスト公開鍵	このコマンドは，NodeManager が HOST_READY になったのを確認した上で発行される．戻り値にはユーザの公開鍵とホストの公開鍵が収められる．
GET_STATUS	N/A	NodeManager ステイタス	NodeManager のステイタスを取得する．ステイタスはノーティフィケーションで非同期に通知されるため必要性はあまりない．
SET_KEYS	known_hosts, authorized_keys, hostfile	N/A	CRM が GRC と通信して取得したグローバルなホスト集合のリストと，各ホストのキー，IP アドレス，公開鍵情報を与える．NodeManager はこの情報を仮想環境にセットする
RUN	N/A	N/A	仮想環境をスタートする
EXEC	実行ファイルパス，引数，環境変数	N/A	仮想環境内でプロセスを起動する．NodeManager の状態が RUNNING の状態でのみ有効
SHUT_DOWN	N/A	N/A	仮想環境をシャットダウンする

実行フレームワークのプロトタイプ実装を通して，設計の有効性を確認した．

今後の課題としては下記が挙げられる．

- **実装** 今回示した実装はアプリケーション実行フレームワーク部分のみを抽出したプロトタイプである．これを我々が構築中のミドルウェアに統合する必要がある．
- **実環境での実験** 上述の実装を行った上で，実際の広域分散環境上で実行を行い，アプリケーション実行フレームワークの有効性を確認する必要がある．

謝　辞

本研究の一部は，情報通信研究機構 (NICT) の委託研究「新世代ネットワークサービス基盤構築技術に関する研究開発」により実施した．

参 考 文 献

1) Takefusa, A., et al. : G-lambda: Coordination of a Grid Scheduler and Lambda Path Service over GMPLS, *Future Generation Computing Systems*, Vol. 22(2006), pp. 868–875 (2006).
2) Kudoh, T.: Grid computing and a role of photonic networks, *Proc. of Network Architectures, Management, and Applications VI(APOC 2008), Proc. SPIE, Vol. 7137* (2008).
3) Thorpe, S. R., et al.: G-lambda and EnLIGHTened: Wrapped In Middleware Coallocating Compute and Network Resources Accross Japan and the US, *Proc. GridNets2007* (2007).
4) Nakada, H., Takefusa, A., Ookubo, K., Kudoh, T., Tanaka, Y. and Sekiguchi, S.: An Advance Reservation-Based Computation Resource Manager for Global Scheduling, *Third Workshop on Grid Computing and Applications*, pp. 3–14 (2007).
5) 岡崎史裕, 工藤知宏, 中田秀基, 竹房あつ子: 経路が動的に接続/解放されるネットワークにおけるユーザ単位の経路切替手法, 情報処理学会研究報告　2007-HPC-111 (2007).
6) Nakada, H., Tanaka, Y., Matsuoka, S. and Sekiguchi, S.: *Grid Computing: Making the Global Infrastructure a Reality*, John Wiley & Sons Ltd, chapter Ninf-G: a GridRPC system on the Globus toolkit, pp. 625–638 (2003).
7) Matsuda, M., Kudoh, T., Kodama, Y., Takano, R. and Ishikawa, Y.: The design and implementation of MPI collective operations for clusters in long-and-fast networks, *Cluster Computing*, Vol. 11, No. 1, pp. 45–55 (2008).
8) Linux-VServer, http://linux-vserver.org.
9) OpenVZ Wiki, http://wiki.openvz.org/.
10) Netperf, http://www.netperf.org/.
11) Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauery, R., Pratt, I. and Warfield, A.: Xen and the Art of Virtualization, *SOSP 2003* (2003).
12) Okajima, J.: Aufs - Another UnionFS, http://aufs.sourceforge.net/.
13) JSON.org: The application/json Media Type for JavaScript Object Notation (JSON), RFC 4627 (2006).
14) : PLANETLAB. http://www.planet-lab.org/.
15) Ozaki, R., Nishida, Y. and Nakao, A.: Building a Flexible Overlay Network Testbed with a Hosted Virtual Machine Monitor, Comsys2008 予稿集 (2008).