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

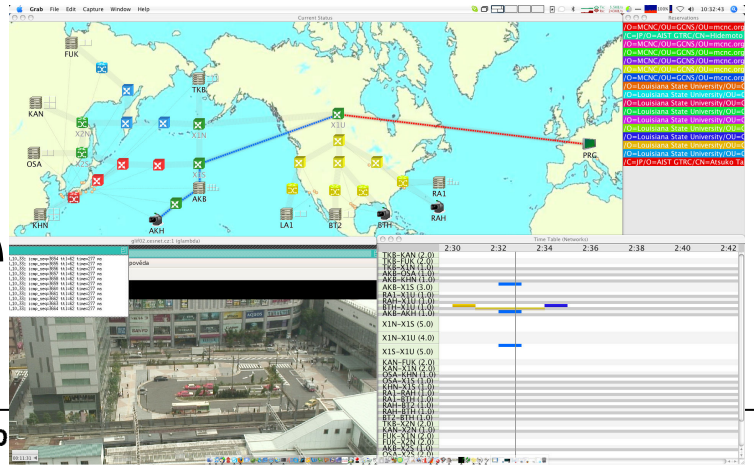
産業技術総合研究所 情報技術研究部門

中田 秀基，高野 了成

竹房 あつ子，工藤 知宏

背景

- 計算資源とネットワーク資源を同時に確保して広域にまたがるアプリケーション実行を行う試み
  - G-lambda 実験
    - 日米欧のサイトを光パスネットワークで接続してアプリケーションを実行
- 複数サイトにまたがるアプリケーション実行の困難さが浮き彫りに
  - 特殊なアプリケーション／ミドルウェアの利用が必須
  - 既存のアプリケーションをそのまま動かすことはできない



背景 (2)

- なぜアプリケーション実行が困難なのか？
  - ネットワークが非対称
    - NAT/Private IP
    - G-lambdaではこれは問題にならない
      - ネットワークもプロビジョンするため
  - 実行に必要な情報が実行時まで決まらない
    - ネットワークを動的に取得する場合のIPアドレスなど
  - 各サイトが不均質
    - 各サイトの管理者は異なる
  - 多くのアプリケーションに共通する不備
    - マルチホームネットワークへの不対応

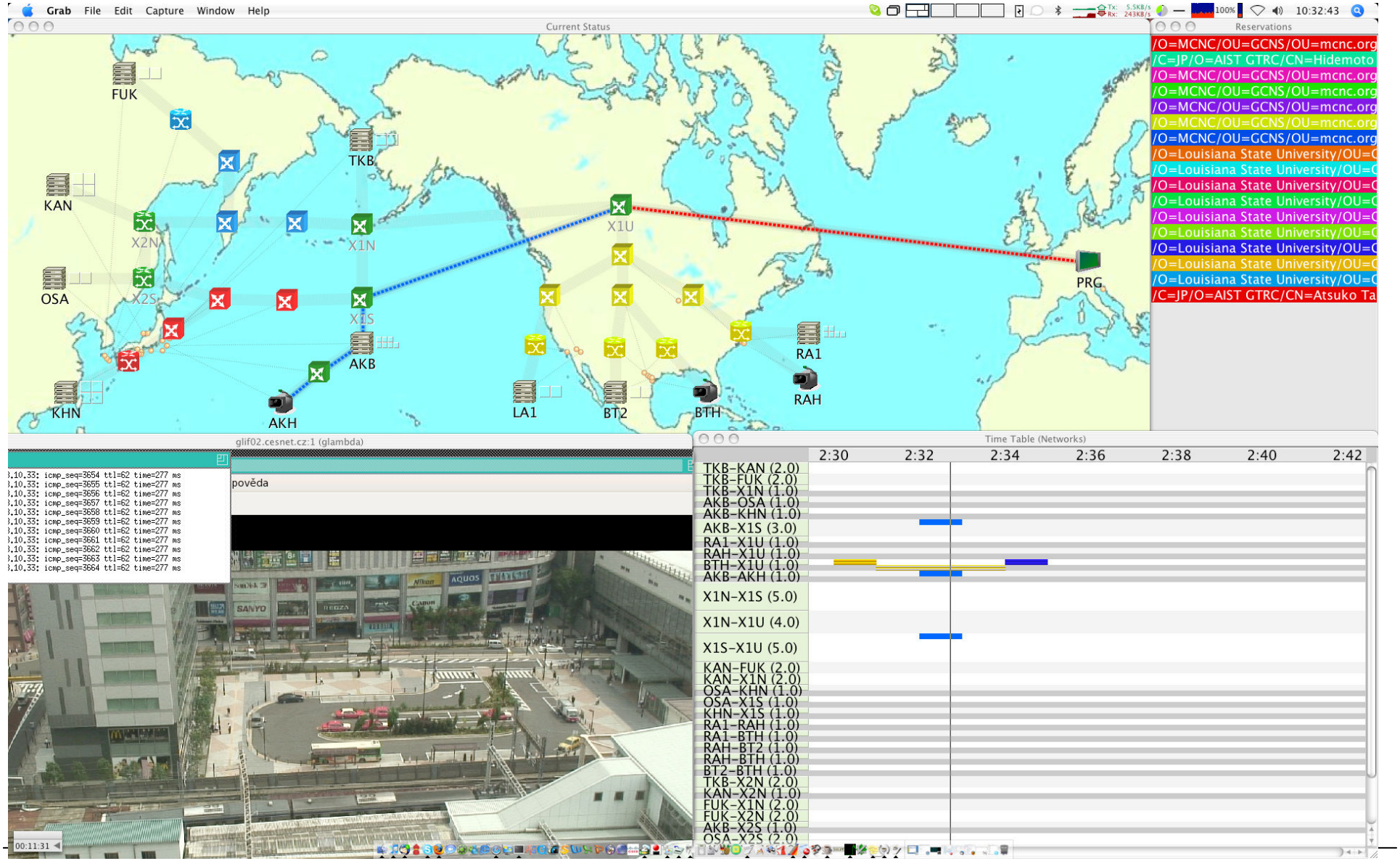
本研究の目的と成果

- 計算機とネットワークを同時に確保するグリッドにおいて，任意のアプリケーションが容易に実行できる環境を実現する
  - アプリケーション実行環境のアーキテクチャを提示
  - Linux-VServerとaufsを用いたプロトタイプ実装によりアーキテクチャの有効性を確認

発表の概要

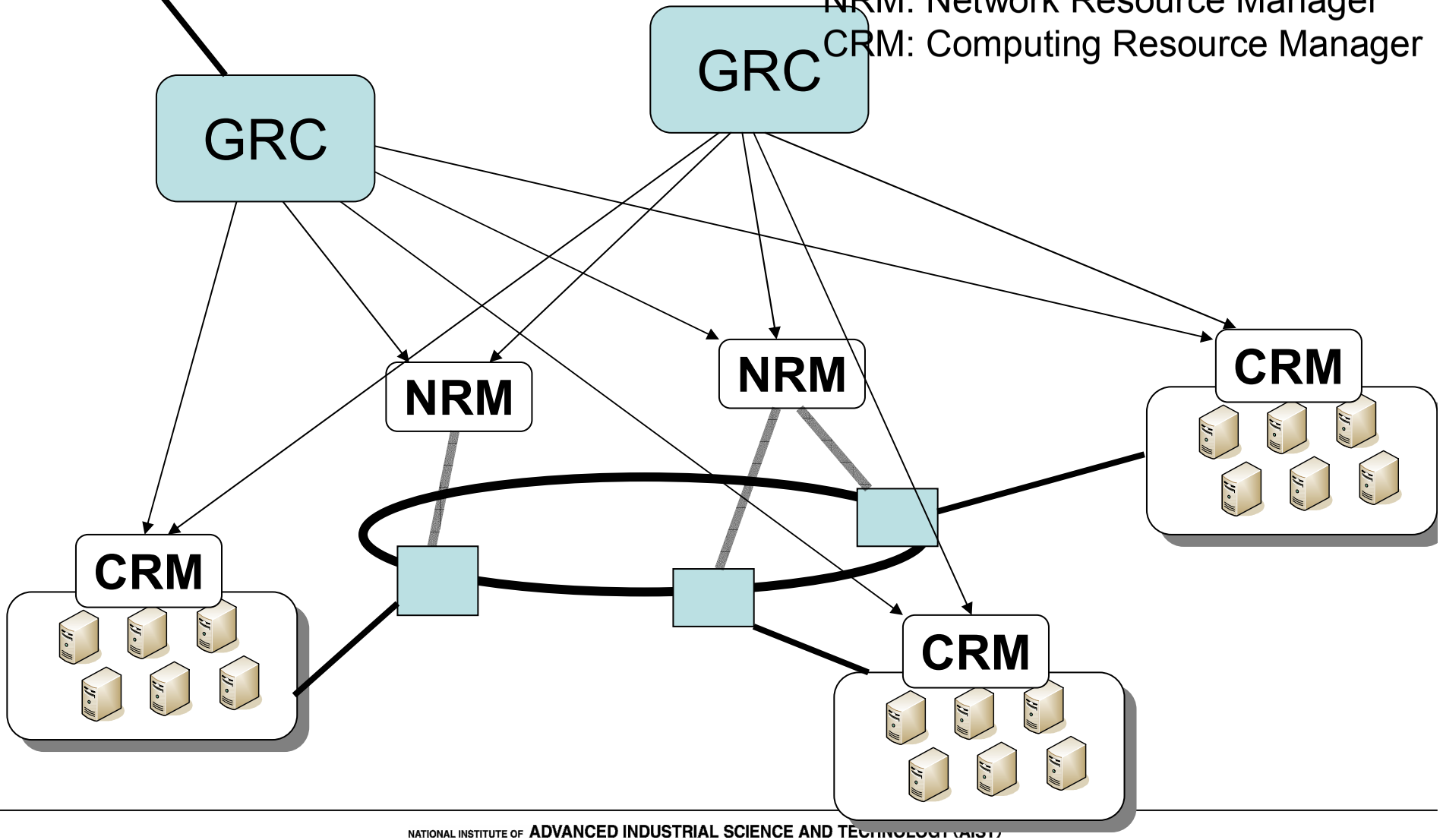
- アプリケーション実行フレームワークの必要性
- 要件の整理と設計
- 実装
- 関連研究
- おわりに

G-lambda実験


Grab File Edit Capture Window Help
10:32:43
Current Status
FUK
TKB
KAN
X2N
X1N
X1U
OSA
X2S
X1S
AKB
PRG
RA1
RAH
KHN
AKH
LA1
BT2
BTH
Reservations
/O=MCNC/OU=GCNS/OU=mcnc.org
/C=JP/O=AIST GTRC/CN=Hidemoto
/O=Louisiana State University/OU=C
/C=JP/O=AIST GTRC/CN=Atsuko Ta
glif02.cesnet.cz:1 (glambda)
pověda
Time Table (Networks)
2:30
2:32
2:34
2:36
2:38
2:40
2:42
TKB-KAN (2.0)
TKB-FUK (2.0)
TKB-X1N (1.0)
AKB-OSA (1.0)
AKB-KHN (1.0)
AKB-X1S (3.0)
RA1-X1U (1.0)
RAH-X1U (1.0)
BTH-X1U (1.0)
AKB-AKH (1.0)
X1N-X1S (5.0)
X1N-X1U (4.0)
X1S-X1U (5.0)
KAN-FUK (2.0)
KAN-X1N (2.0)
OSA-KHN (1.0)
OSA-X1S (1.0)
KHN-X1S (1.0)
RA1-RAH (1.0)
RA1-BTH (1.0)
RAH-BT2 (1.0)
RAH-BTH (1.0)
BT2-BTH (1.0)
TKB-X2N (2.0)
KAN-X2N (1.0)
FUK-X1N (2.0)
FUK-X2N (2.0)
AKB-X2S (1.0)

GridARS アーキテクチャ

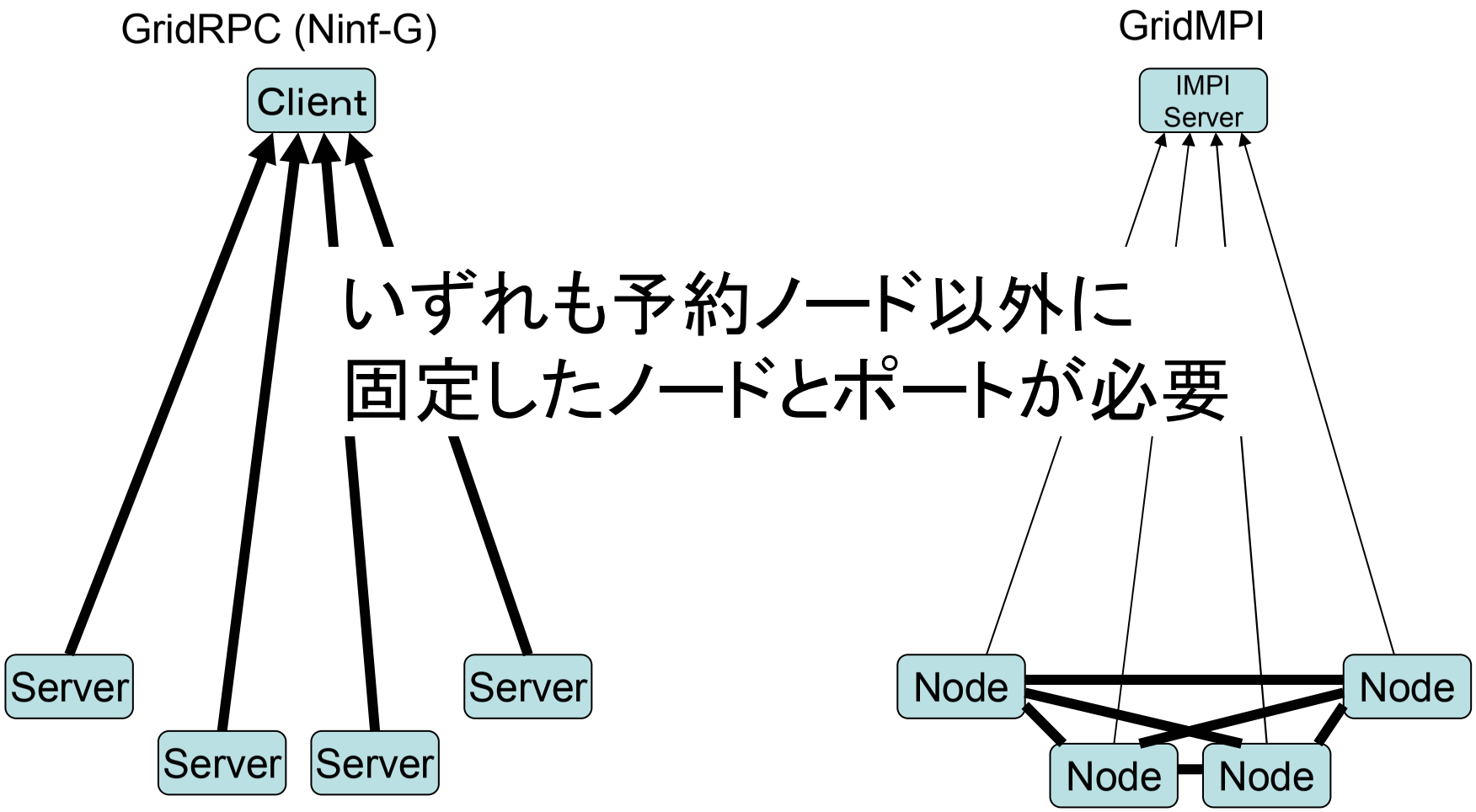
GRC: Global Resource Coordinator
NRM: Network Resource Manager
CRM: Computing Resource Manager



Lessons learned

- IPアドレスの共有がむずかしい
  - 動的にCRMのレイヤでローカルに決定されるため
  - 衝突しないような制御も必要
- 各サイトの設定は煩雑
  - ssh keyの分配，ユーザ名の違いなど
  - authorized_keys, known_hosts の登録
- アプリケーションに不備も
  - 複数のインターフェイスを想定していない
  - 適当なネットワークアドレスを勝手に使ってしまう
    - 性能低下

GridRPC/GridMPIの実行



いずれも予約ノード以外に
固定したノードとポートが必要

アプリケーション実行環境の設計

- 目的：
  - 既存の並列アプリケーションを変更なしに動作させることができる環境の構築
- 前提：
  - 並列アプリケーションの各サブジョブはsshを用いて起動される
    - ホスト名（IPアドレス）はファイルから取得
  - アプリケーションはマルチホームネットワークに未対応
  - 各サイトはそれぞれ使用可能なIPアドレスの領域を持つ．この領域は相互に重なっている可能性がある．

システムへの要請

- すべてのノードのIPアドレスのリストがすべてのノードに配備されている
- すべてのノードのsshユーザ公開鍵，ホスト公開鍵が交換されている
- すべてのノードのIPアドレスは重複していてはならない
  - Private IPアドレスを用いる場合これは自明ではない
- 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
  - マルチホーム未対応アプリケーションへの対応

設計の概要

- 実行環境を整えるパイロットジョブ (NodeManager) を利用
  - ローカルバッチキューイングシステムから直接ユーザジョブを実行せず，パイロットジョブを起動
  - パイロットジョブが実行環境を整備，ユーザジョブを実行
    - IPアドレス，鍵の交換など
- 実行環境には仮想化環境を利用
  - ユーザジョブにみせるネットワークインターフェイスを制御
  - ユーザのホスト環境を変更せず，公開鍵の変更が可能

IPアドレスの決定と収集分配

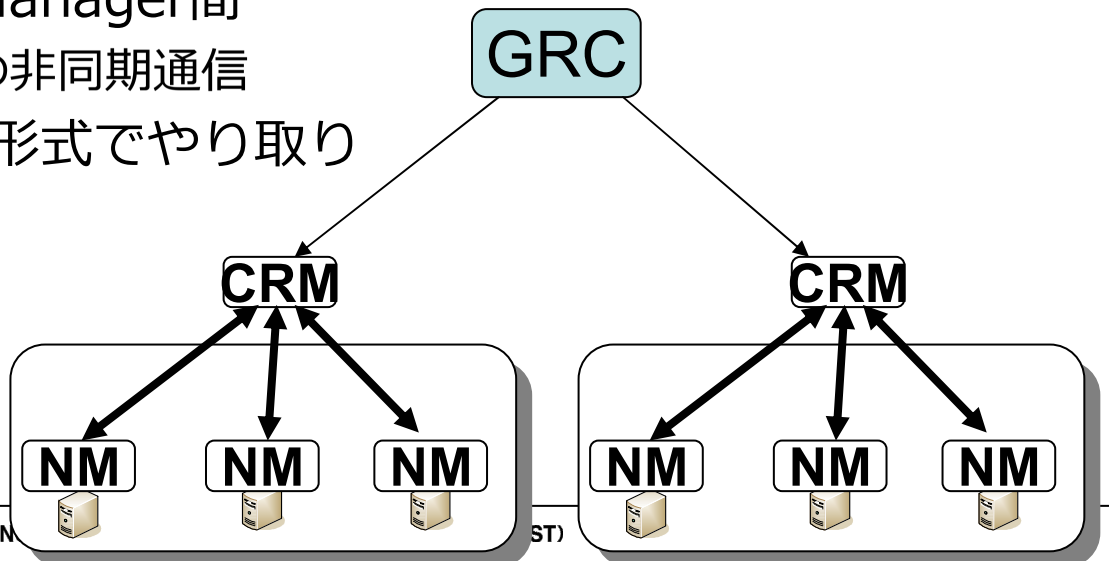
- 事前に，各ノードで利用可能なIPアドレスレンジの集合を公開
- 予約時に，GRCは各サイトで利用するアドレスレンジを決定，CRMに指定
- 実行開始時
  - CRMは利用するIPアドレスを決定，各NodeManagerに対して指定
  - CRMは利用するIPアドレスをGRCに報告
  - GRCはすべてのサイトのIPアドレスをまとめたリストをCRMに返送
  - CRMはそれをNodeManagerに返送

sshユーザ，ホスト公開鍵の交換

- NodeManagerが実行環境構築の一環としてssh ホスト鍵ペア，ユーザ鍵ペアを新たに生成
- 実行時
  - CRMが公開鍵をすべてのNodeManagerから収集，GRCに送信
  - GRCは，すべてのCRMからの情報を集計してCRM経由で分配
    - authorized_keys
    - known_hosts

プロトタイプ実装

- GRC/CRM/NodeManager から構成されるシステム全体のモックアップ
  - フレームワーク構成の妥当性を確認
  - GRCモック/CRMモックはJavaでプロトタイプ
    - 通信はRMIを用いた同期通信
  - NodeManagerはPythonで記述
    - CRMモック/NodeManager間
      - テキストベースの非同期通信
    - 構造データはJSON形式でやり取り



仮想化層の選択

- 計算機仮想化
  - いわゆる'VM'
  - ホストとゲストは完全に分離
  - 負荷は大きい -メモリ，I/O速度など
  - Ex. VMware, Xen, KVM
- OS仮想化
  - OSレベルのサンドボックス
  - ホストとゲストはカーネルを共有
  - 軽量.
  - Ex. Jail, Solaris zone, Linux-VServer, OpenVZ

Linux向けOS仮想化機構

- Linux-VServer
  - PlanetLabで使用されている
  - ホスト・ゲスト間のネットワークの分離が不完全
    - ホストがanyでlistenするとゲストアドレス宛のconnectがホストに捕捉されてしまう
- OpenVZ
  - Parallels社の商用製品Virtuozzoのオープンソフト版
  - ネットワークの分離はより完全
    - パケットをホストがルーティングしている

ネットワーク速度比較

- 1GbEと10GbE で測定
  - 1GbE –Broadcom BCM5722, 10GbE-Myricom Myri-10G
- 使用バージョンは以下のとおり
  - Vanilla -2.6.27-9 server
  - VServer- 2.6.27-10-vz2.3.0.36.2
  - OpenVZ- 2.6.27.10-openvz
  - Xen -3.3, Dom0 2.6.26.1

1GbE [UDP/TCP]

	Mbps	送信側CPU	受信側CPU
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

ネットワーク速度比較

- 1GbE ではどの仮想化機構も十分な性
- 10GbEでは OpenVZ, Xenは性能が十
  - Xen, OpenVZはCPU負荷が高い

測定方法が異なるので
あまり参考にならない.

上3つは8コアで100.0
Xenのみ1コアで100.0

送信側の
CPUが飽和

（12.5 = 1/8）

[UDP/TCP]

	Mbps	送信側CPU	受信側CPU
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

NodeManagerの実装

- Pythonで記述
- 実効uid rootで動作
  - VServerの操作に必須
  - 通常時は実uid nobodyで動作，必要時にrootにseteuidする
- ユーザ権限で動作するLauncherプロセスからsudo で起動
- Launcherが停止すると，自動的に停止
  - バッチキューイングシステムがNodeManagerを間接的に制御することが可能.

NodeManagerに対するコマンド

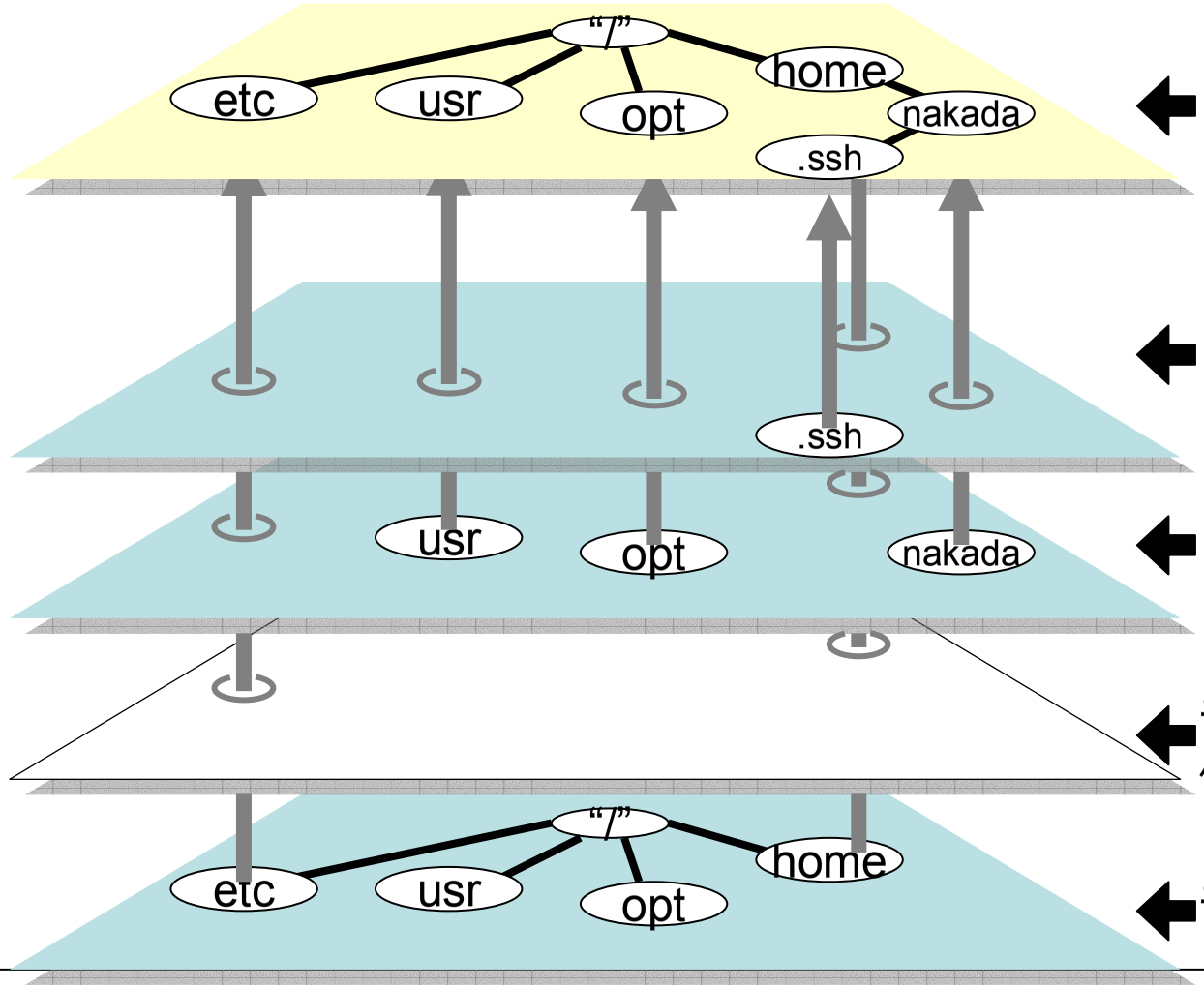
- INIT
  - IpAddressなど必要な情報を与えて仮想環境をセットアップ
- GET_KEYS
  - ユーザ，ホストの公開鍵を取得
- SET_KEYS
  - Hostfile, kownhosts, authorized_keysを設定
- RUN
  - 仮想環境の起動
- EXEC
  - 仮想環境内でユーザのジョブを起動
- SHUT_DOWN
  - 仮想環境を停止
- GET_STATUS
  - 現在のNodeManagerのステイタスを取得.
  - ステイタスは適宜Notify されるので，基本的には不要

仮想化環境内ファイルシステムの構成

- ホスト環境と共有しながら隔離された環境が必要
  - ホスト環境でアプリケーションをテスト，そのまま実行
  - .sshはホストと分離

- aufs(Another UniFS)を使用
  - スタッカブルファイルシステムの1実装
- バインドマウントを援用
  - Mount –t bind

ディレクトリツリーの構成



完成した
ディレクトリツリー

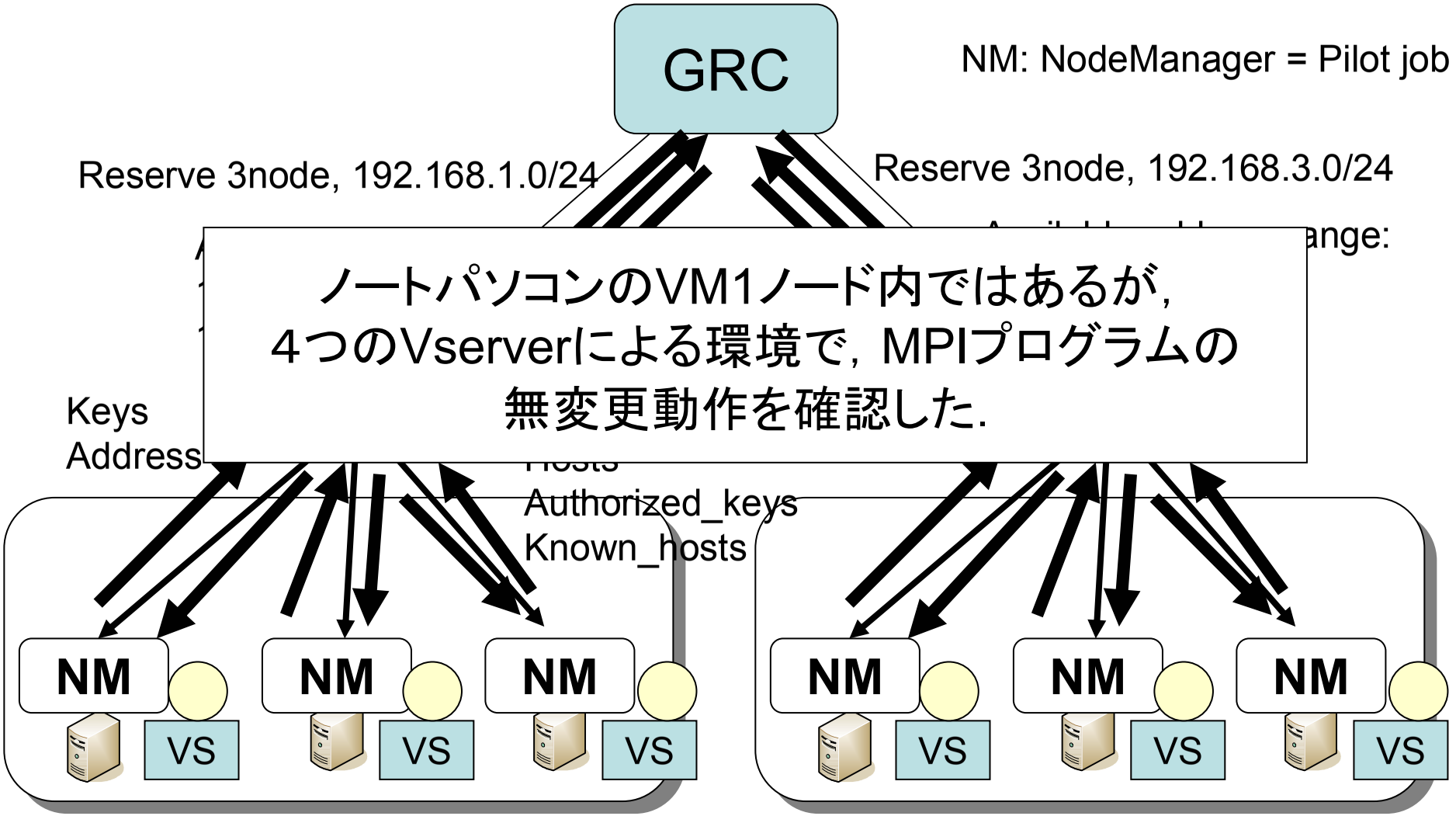
ユーザの.sshを保護する
ために空ディレクトリを
バインドマウント

バインドマウントで
ホストディレクトリの
一部をマウント

テンプレート
保護用の空ツリー

テンプレート

システムの動作


GRC
NM: NodeManager = Pilot job
Reserve 3node, 192.168.1.0/24
Reserve 3node, 192.168.3.0/24
ノートパソコンのVM1ノード内ではあるが,
4つのVserverによる環境で, MPIプログラムの
無変更動作を確認した.
Keys
Address
Authorized_keys
Known_hosts
NM
NM
NM
NM
NM
NM
VS
VS
VS
VS
VS
VS

関連研究

- PlanetLab
  - 広域ネットワークのテストベッド
  - Linux-Vserverを用いてユーザに環境を提供
  - 比較的長期間の貸与
  - 各ユーザ環境は実環境から隔離

- 尾崎ら[comsys'08]
  - NICT AKARIプロジェクト
  - KVMを利用
    - 仮想化環境でカーネルを入れ替えることを重視
    - 1Gbps程度までは対応可能

おわりに

- まとめ
  - 計算機とネットワークを同時に確保するグリッドにおいて，任意のアプリケーションが容易に実行できる環境を実現するためのアーキテクチャを提示
  - プロトタイプ実装を行いアーキテクチャの有効性を確認

- 今後の課題
  - g-lambdaの各コンポーネントに実装
  - 実験
    - 上記を行ったうえで，広域の実環境で実験を行い有効性を確認

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.