多拠点分散環境 (InTrigger) における計算支援ソフトウェアの検証

柴田 剛志† 斎藤 秀雄† 頓 楠†† 横山 大作† 高橋 慧† 弘中 健† 澤井 省吾†
鴨志田 良和† 田浦 健次朗†

† 東京大学情報理工学系研究科電子情報工学専攻
{shibata, h_saito, yokoyama, key, kenny,s1s5, kamo,
tau}@logos.ic.i.u-tokyo.ac.jp
†† 東京大学情報理工学系研究科コンピュータ科学専攻
dunnan@yl.is.s.u-tokyo.ac.jp

InTrigger は, 数百台（現状）のコモディティな計算機で構成された計算機環境であり, 複数のクラスタが，日本の各地にある大学や研究機関に分散して配備されている．本稿では, gxp, vgxp, dds, nicer といった, 本研究室で開発され, InTrigger 環境に提供されている分散並列計算用ソフトウェアやライブラリの概説する．さらに, sshfs, MC-MPI, hadoop などの, 大規模分散計算のための支援システムとして有望なソフトについて, 複数クラスタ環境である InTrigger 環境において, 動作テスト, デバックなどを行ったので, それらについて解説する．

1 はじめに

InTrigger は, 科学研究費補助金特定領域研究「情報爆発に対応する新 IT 基盤研究プラットフォームの構築」で構築している大規模分散計算機環境である．本プロジェクトでは国内の様々な教育・研究機関にクラスタを設置し，6 年間で 20-30 拠点 1000 コア以上の環境を構築することを目指している．

IT 基盤研究のための分散計算機環境の構築は，フランスの Grid'5000 [4] やオランダの DAS-3 [2] など，他にもある．これらの試みに対して，InTrigger には以下のような特徴がある．

柔軟な構成変化を考慮した管理 遠隔地にある拠点の管理や再インストールが容易にでき，システムソフトウェアの研究などが行いやすい．

オープンな管理コミュニティー 拠点に阻まれない管理コミュニティーを作成し，研究者自ら運用方針を決めていく．

拠点数の多さ 20-30 という拠点数は，Grid'5000 や DAS-3 と比べても，多い．

本稿では，まず 2 章で各拠点の構成や拠点間のネットワークなど, 現状での InTrigger 環境の概要を説明する．また, InTrigger のような多拠点に渡る分散計算機環境の構成を柔軟に変化させながら管理する方法, および多数のユーザーが計算資源を共有するためのルールについて説明する．3 章では, 分散環境を使いこなすために本環境で提供されているツールおよびライブラリをいくつか紹介する．これらは, 本研究室で開発されたものである．また，4 章では InTrigger のような大規模広域環境を構築して初めて行える実験があるということを示し，そのような実験を行うことによって得られた知見について述べる．最後に, 5 章でまとめと今後の展望を述べる．

表 2: ping で測定したクラスタ間の RTT(ms)

	chiba	hongo	imade, kyoto	okubo	suzuk
chiba	-	6.2	18	6.3	8.0
hongo	-	-	12	2.7	1.6
imade, kyoto	-	-	-	11	12
okubo	-	-	-	-	4.3
suzuk	-	-	-	-	-

2 InTrigger 環境の概要

本プロジェクトでは 6 年間で 20-30 拠点 1000 コア以上の環境を構築する予定であるが，現在は 3 年目に入ったところで，6 拠点 514 コアが導入済みであ

表 1: 各クラスタの仕様

サイト名	設置機関	CPU	ノード数 (コア数)	メモリ	ディスク (ローカル)	ディスク (共有)	ネットワーク
chiba	国立情報学研究所	Pentium M 1.86GHz Core2 Duo 2.13GHz	70 (70) 58 (116)	1GB 4GB	300GB 500GB	9TB	グローバル
hongo	東京大学	Pentium M 1.86GHz Core2 Duo 2.13GHz	70 (70) 14 (28)	1GB 4GB	70GB 500GB	2TB	グローバル
imade	京都大学	Core2 Duo 2.13GHz	30 (60)	4GB	500GB	9TB	プライベート
kyoto	京都大学	Core2 Duo 2.13GHz	35 (70)	4GB	500GB	9TB	プライベート
okubo	早稲田大学	Core2 Duo 2.13GHz	14 (28)	4GB	500GB	9TB	グローバル
suzuk	東京工業大学	Core2 Duo 2.13GHz	36 (72)	4GB	500GB	9TB	グローバル
合計			327 (514)	888GB	119TB	47TB	

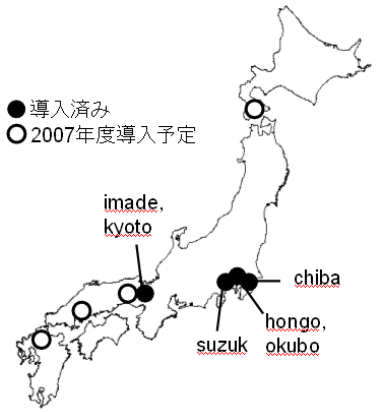


図 1: 導入済み及び 2007 年度導入予定のクラスタの位置

表 3: iperf で測定したクラスタ間のバンド幅 (Mbps)

	chiba	hongo	imade, kyoto	okubo	suzuk
chiba	-	630	93	83	790
hongo	780	-	94	83	820
imade, kyoto	33	46	-	26	51
okubo	81	92	42	-	79
suzuk	66	580	95	78	-

る．図 1 にこれまでに導入済みのクラスタ及び今年度導入予定のクラスタの位置を示す．また，表 1 に各クラスタの仕様を示す．ネットワークの欄が「グローバル」となっているクラスタは全ノードがグローバル IP を持っている．一方,「プライベート」となっているクラスタはファイルサーバ及びいくつかの計算ノードのみグローバル IP を持っており，残りの計算ノードはプライベート IP しか持っていない．

表 2 にクラスタ間の RTT (Round-Trip Time) を示す．現時点では okubo と imade･kyoto の間の RTT が 18ms で最も長いが，今年度北海道と九州にクラスタを導入した後はさらに広域な環境になる予定である．また，表 3 にクラスタ間のバンド幅を示す．全拠点が SINET [7] 経由で 1Gbps 以上の回線で接続されているので，測定されたバンド幅が数十 Mbps しかなかったところに関してはネットワークの設定を確認する必要がある．

2.1 柔軟な構成変化を考慮した管理

様々な拠点に分散したクラスタ計算機の管理を行うとき，効率的に作業を行うためにはなるべく多くの作業をネットワーク経由で遠隔地から行うことができる必要がある．また，システムソフトウェアの研究では様々なソフトウェアをインストールしたり OS の設定を書き換えるような実験を行ったりする場合があるが，このようなとき，ソフトウェアパッケージをインストールするのと同じような感覚で気軽にクラスタ内の各ノードの OS を再インストールできると便利である．

InTrigger 環境ではこれらのことを実現するために，ネットワーク経由で OS のインストールを行うためのツールである Lucie [12] を拡張した．Lucie は Debian の FAI (Fully Automatic Installation) [3] を基に作成された自動インストーラであり，下記の手順でイ

ンストールを行う．

(1) ノードが PXE ブートにより起動

(2) インストール用サーバからインストール用カーネルをダンウロード

(3) ルートファイルシステムを NFS マウント

(4) イントールの実行

元の Lucie ではすべてのノードのインストールが完了するまで待機して，自動インストールを行う設定を解除した後に，各ノードの端末を操作して再起動を指示することが必要であるが，これでは遠隔インストールには向かない．そこで我々は，ステップ (4) の後に次のようなステップを追加することによってインストールが完全に自動的に行われるようにした．

(5) インストール用サーバにインストールの完了を通知

(6) 再起動

インストール用もしくはインストールされるカーネルなどに問題があると自動インストールが途中で止まってしまうが，そのような場合には IPMI (Intelligent Platform Management Interface) でトラブルシュートを行う．IPMI を用いるとノードの電源をリセットすることができ，また，IPMI の SOL (Serial Over LAN) を用いるとコンソール表示を遠隔ノードにリダイレクトすることができる．

また，元の Lucie はステップ (4) で OS だけではなくすべてのソフトウェアをインストールするが，ルートファイルシステムを NFS マウントした状態でのトラブルシュートは困難である．そこで我々は，必要最低限のソフトウェア以外はステップ (6) の後，ノードがローカルディスクから起動した状態でインストールするようにした．再起動後にインストールするソフトウェアの設定はスクリプトで行い，スクリプトを Subversion で管理することによって全ノードに共通のソフトウェアをインストールする．具体的には，Subversion にスクリプトをコミットしておくとノードの再インストール時に自動的にそれがチェックアウトされ，実行される．

2.2 計算資源利用ルールとその運用

InTrigger では様々なジョブが実行されることを想定している．大きな計算資源を利用して，今まで困難だった巨大な処理を行うジョブや，時間当たり高い性能を達成したいジョブ，並列プログラムの開発・テスト実行も行われる．

これらのジョブが無秩序に実行されると，一時的に計算資源を占有することが困難になってしまう．一方で既存の多くのクラスタのようにバッチスケジューラの利用を義務付けると，小さなジョブも毎回ジョブキューを通さなければならず，煩雑である．また，プロセス起動の方法をバッチキュー経由に制限すると，ミドルウェアの開発などが制限されてしまう．

そこで InTrigger では，資源利用ルールを導入し，ルールを守る範囲内ではプロセスの起動方法を限定しないことにした．以下，これらについて詳しく説明する．

2.2.1 計算資源利用ルール

利便性と様々なミドルウェアの利用・開発のために，本 InTrigger プラットホームではプロセス起動の手続きは限定しない．ユーザはバッチキューに限定されず，対話式シェル・独自のミドルウェアなど様々な方法でプロセスを起動することができる．

一方で短時間計算資源を占有したいユーザのためにバッチスケジューラである TORQUE バッチスケジュラーを設置した．このスケジュラー経由で起動されたプロセスには優先権を与えるものとし，それ以外のプロセスを非優先プロセスと呼ぶことにする．この優先権について二つのルールが設定し，これに違反することを迷惑行為と定義する．

ルール 1: 優先権のあるプロセスの邪魔をしない
TORQUE のジョブキューを通して実行したプロセスに優先権があるとし，ジョブキューを通さずに実行したプロセスがある程度以上 CPU を利用した場合は迷惑行為とみなす．具体的には非優先プロセスが利用できる CPU 時間は (CPU 数?キュー経由のプロセス数) $\cdot$ 100% とし，これを超えて利用してはならないものとする

ルール 2: 優先権を濫用しない キューで他のユーザのジョブが待たされているときに，優先プロセスを制限量を超えて走らせ続けてはいけないとする．制限量に関しては (使用プロセッサ数 x 使用時間) で評価するのが自然だが，本環境では並列実行を推奨するために ($\sqrt{\text{使用プロセッサ数}} \cdot$ 使用時間) を用いるものとする．制限値としては，

全プロセッサを 1 時間占有できるような値に設定する．例えば 512 プロセッサで 1 時間と設定した場合，128 プロセッサの場合 2 時間，1 プロセッサの場合約 1 日となる．

2.2.2 計算資源利用の運用

各ユーザは前節で説明した二つのルールを遵守するものとする．ユーザにルールを守らせるための仕組みとしては，各ノードの情報を定期的にモニタリングすることにより集計し，迷惑行為を検出してユーザ毎の週報を作成する．迷惑行為は数値化した「ポイント」としての集計する．このポイントの計算方法は以下のようになる．

ルール 1: 優先権のあるプロセスの邪魔をしない 非優先プロセスの CPU 利用率が制限を超過しているとき，超過分をその時に動いている非優先プロセスの CPU 利用率で比例配分し，ポイントとする．

ルール 2: 優先権を濫用しない キューを制限を超えて使用した場合，((超過した時間) · (使用プロセッサ数) · (待たせたジョブ数)) をポイントとする

迷惑行為さえ行わなければどのようにジョブを実行してもかまわないが，典型的な使い方としては以下のようなものを想定している．資源を占有したいジョブは，キューを用いることで実行させる．またデバッグ用の小規模な実行などは CPU をあまり使わないので，対話的に実行しても迷惑行為とみなされることはない．また長時間空き資源を利用したいジョブは，定期的に CPU 使用率を監視して適宜停止・再開する必要が生じるが，これを簡単に行うツールとして，nicer というソフトウェアを準備した．これについて次節で説明する．

2.2.3 Nicer

nicer は nice コマンドの拡張で，計算資源の利用状況に応じてプロセスを一時停止させたり再開させる．TORQUE のジョブキューを通して実行したプロセスが現れたらプロセスを一時停止させ，消えたらプロセスを再開させることによって，迷惑行為を行うことなく空き CPU 時間を有効利用する．nicer 同士の公平性を保つために，一定時間 (標準では 10 分) 実行後自発的に停止し，もし他のプロセスが立ち上がらないようであれば再開する．これにより，複数の nicer プロセス間で公平に資源が分配される．

3 本研究室で開発され提供されているツール

本章では，分散環境を使いこなすために InTrigger で提供されているツールをいくつか紹介する．従来から並列分散環境を対象としたツール，ミドルウェアは数多く存在する．しかし，その多くは単一クラスタ環境を対象としている．我々が構築した InTrigger は，既に 6 拠点ものクラスタをまたがり，今後も増加していくことを考慮すると，新たな広域分散環境に対応したツール，ミドルウェアが必要である．

3.1 GXP

分散環境を効率的に扱うためには多くの計算機への素早いコマンド投入やこれらの計算機を協調動作させる分散アプリケーションを簡単に記述できることが重要になる．また，これらを複雑な設定を行わずに可能にすることが求められる．Grid eXPlorer(GXP) [9] はコマンドラインインタフェースの並列シェルで，複数拠点にまたがる分散環境で効率的に利用できるログイン機能とコマンド実行機能が実装されている．

ログイン機能は，少ない設定で現在利用可能なノード群に素早くログインが可能になるように設計されている． あらかじめ各ノードへのログイン方法を指定しておき， ログインコマンドを実行することで指定した計算機に並列にログインを行い， GXP を終了するまでセッションを維持する． 直接ログインできない計算機へも， 一旦ゲートウェイのノードにログインした後にその内部のノードにログインを行うように設定できるため， プライベートネットワークやファイヤーウォールがある環境でも使用することができる．ログイン方法の指定は， 複数の可能なログイン経路を冗長に記述することができ， 故障ノードが存在するなどしてログインできなかった場合は，自動的に他の経路を試す． ssh を使ったログインの他に， TORQUE [10] や Sun Grid Engine を経由してログインすることも可能である． また， インストールを簡単にするため， GXP はログインが成功したノードに対して， 自分自身をコピーする機能を有している．

GXP では，コマンドを送信すると，それがすべてのノードで実行されるという単純なモデルでコマン

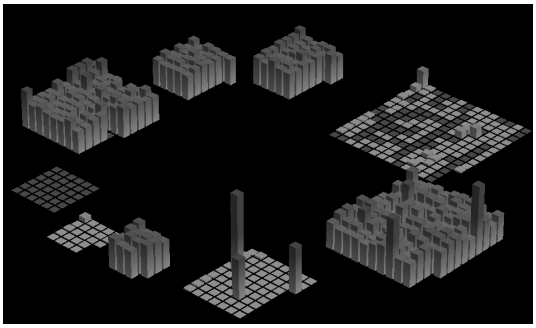


図 2: VGXP のスクリーンショット

ドが実行できる．各ノードで実行されるコマンドへのデータのブロードキャストやコマンドの出力の集約を Unix シェルの入出力のリダイレクトのように記述することができるため，マスター・ワーカー型の並列プログラムを簡単に作ることができる．前回実行したコマンドの終了ステータスによって，次にコマンドを実行するときのノードを選ぶ機能を使うことで，特定のノードのみでコマンドを実行することも可能である．

3.2 VGXP

並列計算環境は，計算機の台数が増えるほど，また分散している拠点の数が増えるほど安定した状態で利用することが困難になる．

不測の問題が発生したときに，現在の計算環境の状態を迅速に把握して対処を行うためにはモニタリングシステムの存在が不可欠となる．

複数の拠点に対応できるモニタリングシステムとして，たとえば Ganglia や Nagios が存在する．これらのツールはノードの生死や監視項目の値があらかじめ設定した範囲にあるかどうかを判定してそれらを表示することができる．しかし，望ましい値の範囲を静的に決定することは難しいし，設定も煩雑である．また，これらのツールは 1 分から数分ごとの値の変化を把握するように設計されているため，短時間のジョブについての動きを監視するには適していない．

Visual GXP(VGXP) [11] は，通信量や監視の負荷を抑えつつ 1 秒から数秒間隔で各監視項目の値を集め，各計算機の状態変化を迅速に把握できるように設計された多拠点分散環境向けのモニタリングシステムである．収集した各計算機の情報はそれぞれを比較しやすいように可視化を行い，特別にしきい値を設定しなくとも異常の可能性を視覚的に判断し，必要に応じてさらに詳しい情報を見ることができる．図 2 にスクリーンショットを示すが，小さい画面に多くの情報を表示できるように，各ノードを 2 次元平面上に配置し，それぞれのリソース利用率を縦棒グラフで表すという，3 次元的な可視化を行っているのが特徴である．収集している情報は，CPU やメモリの使用率，通信量やディスク I/O の量，アクティブなプロセスの CPU 使用率などで，他の情報を収集するように拡張することもできる．また，クライアントのインストールを簡単にするため，クライアントには Java Web Start を利用している．このため，ブラウザからリンクをクリックするだけでアプリケーションのインストールと起動を行うことができる．

3.3 dds

dds は，分散環境上で容易にプログラムを記述できるように開発された，分散オブジェクト指向ライブラリである．このライブラリでは通常のオブジェクトと遠隔に存在しているオブジェクトに対するアクセスを透過的に扱うことができるため，オブジェクトの位置を気にすることなくプログラムを記述することができる．同期的な RMI はもちろん，非同期的な RMI も呼び出すことが可能である．

従来から RMI を透過的に見せるミドルウェアは数多くあった．しかし，それらは大規模の台数で動作することは想定されておらず，例えば一つのオブジェクトの一つの参照につき，スレッドを一つ，ネットワーク接続を一つ，などという設計になっている．また，並列環境を想定したものであっても，参加している計算資源間で，全対全の接続が張れるという仮定がある．これは InTrigger のような台数の大きいでは大きな制約となり，NAT や Firewall にも対応できない．さらに，計算に使う資源の構成を設定ファイルに記述するなど，アプリケーションを書く人に敷居が高くなっている．

dds では，通信遅延を意識したオーバーレイネットワークを構築し，内部でルーティングを行うため，連結であれば通信することが可能である．このため，NAT や Firewall が存在する環境下であっても動作し，大規模な台数になってもスケールする．また，事前に計算に参加する資源などを設定ファイルに指定する必要はなく，動的な資源の追加，削除を許す．

オブジェクトの位置が固定されていると，性能上

非効率であったり，動的なプロセスの増減に対応できないため，dds ではオブジェクトマイグレーションをサポートしている．この機能はすべてのリモートオブジェクトに対して呼び出すことができ，アクセスの多いプロセスにオブジェクトを移動させることで効率的に実行することができる．

また，現時点では明示的に呼び出さない限り実行されないが，分散 GC も実装されている．この分散 GC はスナップショットでとったオブジェクトグラフを元にマークスイープを行う分散 GC であり，参照されていないすべてのオブジェクトを回収することが可能である．

3.4 MC-MPI

Multi-Cluster MPI (MC-MPI)[6] は広域環境用の適応的な MPI ライブラリである．広域環境ではファイアウォールや NAT などによって一部のノードの間の通信が遮断されていることがある (InTrigger の場合 kyoto と imade がこれに該当する)．このような環境で並列計算を行うためには通信できるノードの間で接続を確立し，直接通信できないノードのためにメッセージを中継する必要がある．既存の MPI ライブラリにも MPICH/MADIII [?] のようにこのような中継を行うものがあるが，中継のための設定を手動で行う必要があり，計算環境の規模と複雑さが増すと煩雑になってしまう．これに対して MC-MPI は手動の設定を必要とせず，実行時にノード間の接続性を調べ，たまたま確立することができた接続を用いてルーティングレイヤを構築する．

また，多数のノードで密に接続を確立するとシステムの様々な資源の制限 (e.g., メモリ量，ファイルディスクリプタ数，ステートフルファイアウォールのセッション数) に触れてしまう可能性があるが，MC-MPI は各プロセスが確立する接続の数を制限することによってスケーラビリティを向上させる．この際，遠い (遅延の高い) ノードとの接続を削減し，近い (遅延の低い) ノードとは密に接続を確立することによって通信性能を維持する．ノード間の遅延は起動時にピンポンを行うことによって自動的に取得する．

このように MC-MPI は計算環境に適応するために接続性を調べたりピンポンを行ったりするが，これらのことを局所性を考慮して行うため，高速に起動する．例えば，遅延測定の際には三角不等式に基づいて遠いノードとの遅延を見積もる．遠いノードとの遅延を見積もると，測定回数も減り，時間がかかる測定を回避することもできる．このような工夫の結果，InTrigger の 6 拠点 486 コアでは MC-MPI は 12.5 秒で起動した．

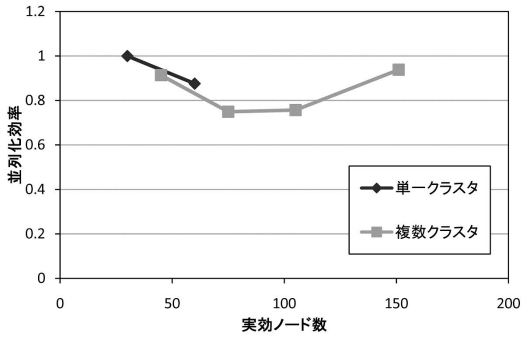


図 3: 数独探索プログラムの実行効率

MC-MPI を用いると広域環境で UPC (Unified Parallel C) のプログラムを実行することもできる．Berkeley UPC [1] ランタイムは MPI を用いて通信を行うことができるので，MPI ライブラリとして MC-MPI を用いることによって，ファイアウォールや NAT がある広域環境で UPC プログラムが実行できるようになる．

4 大規模広域環境における既存ソフトウェアの動作検証

4.1 Ibis, Satin

Ibis[?] は Java を用いた広域分散計算を可能にする計算環境であり，メッセージ通信や Remote Method Invocation などを GRID 環境上で提供している．

この環境上でより高水準な並列プログラミングを可能にする言語処理系として，Satin[?] が実装されている．Satin は，探索問題などに頻繁に現れる divide-and-conquer 型のアルゴリズムを簡単に自動並列化することを目的としており，ユーザが Java プログラムのあるメソッドに対して並列実行可能であると指定すると，Satin がバイトコード変換を行って通信や制御などのための機能を埋め込み，そのメソッドが自動的に分散計算される．lazy task creation に基づいた task stealing による動的負荷分散を実装しており，タスク分割が規則的でないような問題に対しても高い計算効率を実現できる．

「数独」というパズルの可能解を数え上げるプログラムを用いて，InTrigger 上での Satin の動作実験を行った．図 3 に実行効率を示す．用いた計算機構

成は以下の通り．

- hongo クラスタ 30 ノード
- hongo クラスタ 60 ノード
- (hongo, chiba, okubo) = (10, 10, 10)
- (hongo, chiba, okubo) = (30, 20, 10)
- (hongo, chiba, okubo) = (40, 40, 10)
- (hongo, chiba, okubo) = (63, 63, 10)

okubo のみ CPU 構成が異なるため，本プログラムを単一計算機上で性能測定したところ，他クラスタの計算ノードの 2.5 倍の速度であった．このため，グラフの横軸は okubo クラスタの参加計算ノード数を 2.5 倍にした実効ノード数で表示している．グラフの縦軸は，理想的な台数効果を 1 とした場合の実際の台数効果の比率を表している．ただし，単一クラスタ 30 ノードでの実行効率を基準としている．動的負荷分散を行っているため，実行時間にはばらつきがあるが (グラフは 3 回の測定の平均を表す)，複数クラスタを用いた場合でもそれほど効率を下げることなく分散計算が行われていることがわかる．

この実験を通して，以下のような問題点が判明した．

- NAT 環境に対応していない

  Ibis は firewall 等の接続制限のある計算環境においても，ルーティングなどの手段を用いて通信を試みる．しかし，InTrigger のように NAT を含む環境においては，この機構は動作しなかった．エラーログから判断する限り，Ibis の接続を一元管理する nameserver がプライベートアドレスを正しく扱えないようであった．

- 複数のネットワークを持つ計算ノードに対応していない

  InTrigger には，グローバルアドレスと NAT 内のプライベートアドレスを両方とも持つノードが一部存在する．これらのノードは NAT の対象とはなっていないのであるが，Ibis が誤ってプライベートアドレス側に通信用ソケットを作成し，nameserver に接続できず計算に参加できないという状況が確認された．

また，Java1.6 系列では動作が極めて不安定になる，firewall 越えの設定で動作しているときには，java のオプションに java.net.preferIPv4Stack=true を設定してソケットを明示的に IPv4 で使用しないと nameserver が動作しない，といった問題点も明らかになった．

4.2 SSHFS

SSHFS (SSH Filesystem) [8] は，SSH [?] の sftp プロトコルを用いた，FUSE (Filesystem in Userspace) [?] フレームワークで実装したファイルシステムである．

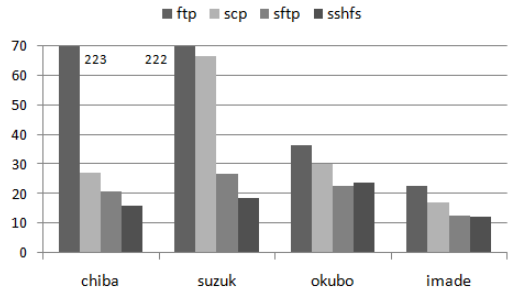
この SSHFS によって，リモートのサーバのディレクリをローカルディレクトリに FUSE でマウントし，リモートにあるファイルにローカルファイルと同様にアクセスすることができる．グリット環境での SSHFS の利便性については，二つの側面がある．まず，マシン間のファイルを共有するには通常サーバ側での設定が必要であるが，SSHFS にとってサーバ側は SSH サーバさえあれば，特別な操作をいらない．一方で，SSHFS は標準な SSH コネクションを用いたので，クラスタ間の NAT や Firewall もセキュア的に渡られる．InTrigger では，SSHFS をデフォルトなパッケージとして全ノードにインストールされ，利用することができる．

InTrigger で SSHFS の性能評価実験を行った．評価実験は hongo と他の拠点 (chiba，suzuk，okubo，imade) の計算ノード間に各転送方法 (ftp，scp，sftp，sshfs) を用いてファイルコピーを行った．実験に用いたソフトウェア環境の構成は表 4 のようになっている．

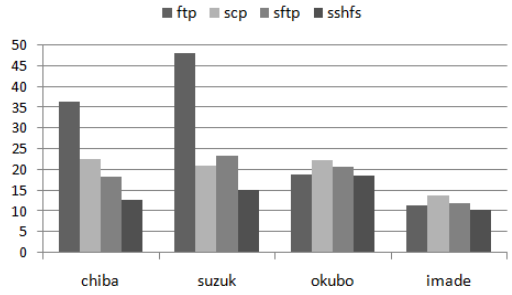
表 4: 実験に用いたソフトウェアの構成

Software	**Version**
vsftp	2.0.3
OpenSSH	4.3p2
SSHFS	1.8
FUSE Library	2.7.0
FUSE Kernel Interface	7.8
Linux Kernel	2.6.18

512MB のファイル一個を転送した時と 1MB のファイル 512 個を転送した時の実験結果をそれぞれ図 4(a)，図 4(b) に示す．SSH プロトコールを用いた scp は, ftp と比較して, 平均で 0.68 倍の性能に抑えられていた．この性能低下は SSH の暗号化オーバーヘッドとみなすことができる．一方，SSHFS は scp よりも 0.83 倍低い性能を表している．これは FUSE がシ



(a) 512MB×1 個ファイルの転送速度の比較



(b) 1MB×512 個ファイルの転送速度の比較

図 4: 各転送方法の転送速度の比較 (Mbps)

ステムコールを頻繁によぶため，カーネルスペースとユーザスペースの切り替えが頻繁におこるためである．

SSHFS の利用にはもう一つの問題点がある．FUSE Library2.5.3(含む) 以降を用いたとき, SSHFS のマウントポイントを NFS (Network File System) に乗せると，NFS サーバに必要以上に大きい負荷をかけることが明らかになった．

4.3 MPICH2

グリッド用のソフトウェアは広域環境で多数のノードにスケールする必要があるが，大規模広域環境におけるソフトウェアの検証は容易ではない．そもそも実験を行うための環境が必要であるが，少数の団体のアドホックなコラボレーションではそこまで大規模で広域な環境を整えることができない．本章では，InTrigger のように多拠点に渡る分散計算機環境を構築することによって初めて行える実験があるということを示し，そのような実験を行うことによって得られた知見について述べる．

例として MPI ライブラリとして広く用いられている MPICH2 [5] の検証を行った．MPICH2 は特にグリッドを意識したライブラリではないが，ファイアウォールや NAT のない環境では動作するはずのものである．バリアを張るだけのプログラムを 1 クラスタ 83 ノード (chiba のみ) と 3 クラスタ 83 ノード (chiba，hongo，okubo) で実行したところ，1 クラスタの場合は毎回問題なく実行できたが，3 クラスタの場合は高い確率でデッドロックを起こしてしまった．

```
# メッセージヘッダを受信
header = recv(8)
size = get_size(header)

# メッセージ本文を受信
while received < size:
  frag = recv(size - received)
  body += frag
  received += len(frag)
```

図 5: バグを含んでいた部分のコード

デッドロックが生じる直接の原因は，バリアを張るための send の中でハングしてしまうプロセスがあるということであったが，実際のバグはそのプロセスとはまったく関係ないところにあった．MPICH2 では，各プロセスは他のプロセスと最初に通信を行うときに初めてそのプロセスとの接続を確立する．その際にリング上に接続されているプロセスマネージャを介してエンドポイントを交換するが，デッドロックの元々の原因はこのプロセスマネージャのバグにあった．

プロセスマネージャのバグを含んでいた部分のコードを図 5 に示す．`recv` システムコールは要求したバイト数より少ないバイト数しか受信せずに返ることがあるためループの中で呼ぶ必要があるが，図 5 のコードではメッセージヘッダを受信する部分にはループがない．ヘッダはわずか 8 バイトであるため，単一のクラスタという低遅延環境では常にまとめて届いたが，3 クラスタを用いた場合はより遅延が高かったため，8 バイトでも分かれて届いてしまうことがあった．その結果，ヘッダの一部しか受信せずに本文の受信に進んでしまうことがあった．

このように InTrigger を用いることによって小規模な環境では生じないバグを見つけることができた．しかし，バグが広遅延環境でしか生じないものであり，また，デッドロックが検出された箇所と実際に

バグがあった箇所の間には多数のプロセスが絡んでいたため，流れを手動で追うことは容易ではなかった．これは，このような大規模分散環境における問題追跡を支援する仕組みの重要性を現している．

4.4 Hadoop

Hadoop は，MapReduce [?] に基づいた分散環境における計算支援ライブラリである．JAVA で書かれたオープンソースで，主に膨大にある Web データの解析や自然言語処理の際に使われる．システムは，主に，Google ファイルシステムに相当する分散ファイルシステムとジョブを処理するための MapReduce の部分の 2 つからなっている．MapReduce の部分はマスターワーカである．採用されている分散ファイルシステム（Hadoop ファイルシステム）は，ファイルを断片化して，複数のノードに耐故障のために重複しておいておくようになっている．

Hadoop では，ジョブの入力ファイルを Hadoop ファイルシステムからとって，MapReduce の結果を Hadoop ファイルシステムに出力するように意図して設計されている．このため，ユーザは，Hadoop ファイルシステムを通してデータをやりとりしなければならない．

一方で，InTrigger のような広域に及ぶ多拠点クラスタ環境では，拠点間ネットワークの遅延，バンド幅がシステム全体のパフォーマンスに与える影響が無視できないと考えられる．そこで本小節では，Hadoop の多拠点環境での動作確認と動作の条件について述べ，サンプルプログラム (ソート) をもちいて，多拠点でノード数を変えて，オーバーヘッドに相当すると考えられる部分をチェックする．

まず，分散ファイルシステムを統括するためのデーモンが一つと，実際のデータの断片を保持するノード全てにデーモンが立っいるため，NAT を間にはさむことはできない．MapReduce 部分についても，Master, Worker それぞれでデーモンが立っている必要があり，同じことがいえる．また, 実際の断片を保持するノードは，分散して重複したデータを持っているとはいえ，多くのノードを簡単にかえることはできないため, 同じ大量のデータを何度も使って実験を行いたい場合は，事実上固定しなければならない．

次に，4 拠点を使った場合のソートプログラムの実行結果を表 5 に示す．あらかじめ，100B 以下程度のランダムなキーとバリューのペアを 2GB 分作成し，Hadoop ファイルシステム上に置いておく．使った拠点は，hongo(ho)，suzuk(su)，chiba(ch) および okubo(ok) であり，それらの間のネットワークのバンド幅，遅延等は 2 節で述べたとおりである．

表 5: Hadoop を用いた 2GB ランダム生成ファイルのソートの実行時間

ノード (総数)	4	72
ノード (内訳)	ho × 1, su × 1, ch × 1, ok × 1	ho × 24, su × 12, ch × 24, ok × 12
総実行時間 (msec)	2950	2370
実行時間 (内訳)	map : 1390, red : 240, fs : 1320	map : 320, red : 750, fs : 1300

そもそも Hadoop 自身が Map 関数の結果を Reduce 関数に渡す際にソートするように作られているので，ソートのサンプルプログラムは，Map 関数，Reduce 関数は入力をほぼそのまま結果として出力すればよいだけである．そのため，いわば空回しと言ってよく，Hadoop ファイルシステムへのアクセスおよび MapReduce 自身のオーバーヘッドと考えられる．

結果 (表 5) からわかるように，処理が終わるまでの総時間は，Worker の数をふやすとわずかに減少した．また，それぞれの処理の過程にかかった時間を見ると，違いがある．表中の map は，全ての Map が終わり，その結果のデータが Reduce を実行するノードに転送されるまでの時間である．red はソートおよび Reduce にかかった時間，fs は結果を分散ファイルシステムに書き込むのにかかった時間である．まず，Worker の数が増えれば，ある程度 map 部分に相当する計算時間は減少している．これは，ファイルの断片が分散しておかれているため，同等数程度に分散して処理すればそれだけ帯域を確保することができるためと考えられる．一方で，Reduce 部分に相当する時間は減少していない．これは，最終的にひとつの Worker が結果をひとつの大きなファイルとして，ソート結果を Hadoop ファイルシステム上に書き込むためと考えられる．

Hadoop は，処理が Map と Reduce に分割でき, 処理すべきデータ量がある程度大きく，かつ個々のデータの処理時間が，それを分散ファイルシステムに読み書きするのにかかる時間よりも十分大きい場合に，効果を発揮する分散計算ライブラリであるといえる．

5 おわりに

多拠点クラスタ環境である InTrigger において，いくつかの計算支援ソフトウェアの検証を行った．基

本的に，1 クラスタまたは密につながったクラスタを仮定しているソフトウェアにおいて，思いもよらないバグの存在や，制限が存在することがわかった．例えば，Ibis や Hadoop では，基本的に NAT をはさむことができない．また，MPICH2 では，クラスタ内では事実上おこりえないようなメッセージの分断が今回のような環境では頻繁に起こるため，該当個所を修正する必要があった．また，Hadoop では，分散ファイルシステムとのデータのやりとりが，オーバーヘッドの主要な部分をしめることがわかった．そのため，クラスタ間のネットワークの遅延やバンド幅の制限を考慮することが重要である．

今回，複数の分散計算支援ソフトウェアについて，それぞれの問題点を個別に明らかにし対処するために労力を費したが，新しく使用を計画するたびに同じことを行うと，使うことができるようになるまでの負担が大きくなる．より効率的に今回のような動作確認，ボトルネックの原因追求を行えるような，統一的な方法が求められる．

謝辞

本研究は文部科学省科学研究費補助金特定領域研究「情報爆発に対応する新 IT 基盤研究プラットフォームの構築」の助成を得て行われた．本環境を構築するにあたって，NEC の高宮氏に Lucie の使用方法や修正について支援を頂いた．また，クラスタを設置した各研究室 (京都大学角研究室・中島研究室，早稲田大学山名研究室，東京工業大学合田研究室) の皆様にも協力を頂いた．

参考文献

[1] Berkeley UPC. Online at http://upc.lbl.gov/.

[2] DAS-3. Online at http://www.cs.vu.nl/das3/.

[3] FAI - Fully Automatic Installation. Online at http://www.informatik.uni-koeln.de/fai/.

[4] Grid'5000. Online at http://www.grid5000.fr/.

[5] MPICH2. Online at http://www-unix.mcs.anl.gov/mpi/mpich/.

[6] Hideo Saito and Kenjiro Taura. Locality-aware Connection Management and Rank Assignment for Wide-area MPI. In *Proceedings of the 7th International IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007)*, pages 249–256, 2007.

[7] SINET 学術ネットワーク. Online at http://www.sinet.ad.jp/.

[8] SSH File System. Online at http://fuse.sourceforge.net/sshfs.html.

[9] Kenjiro Taura. GXP: An Interactive Shell for the Grid Environment. In *Proceedings of the 8th International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA 2005)*, pages 59–67, 2005.

[10] TORQUE Resource Manager. Online at http://www.clusterresources.com/pages/products/torque-resource-manager.php.

[11] 鴨志田良和, 金田憲二, 遠藤敏夫, 田浦健次朗, and 近山隆. VGXP: 多数の計算機をリアルタイムに監視・操作するソフト. In **並列/分散/協調処理に関するサマー・ワークショップ** *(SWoPP 2006)*, pages 19–24, 2006.

[12] 高宮安仁, 真鍋篤, and 松岡聡. Lucie: 大規模クラスタに適した高速セットアップ・管理ツール. **情報処理学会論文誌：コンピューティングシステム**, 44(SIG 11 (ACS 3)):79–88, 2003.