Vol.2015-HPC-151 No.4
2015/9/30

多拠点連成アプリケーションを実現するユーザ駆動型・拠点連携システム

實本 英之[4,a)]　小林 泰三[2,b)]　松本 正晴[1,c)]　滝澤 真一朗[3,d)]　三浦 信一[4,e)]

概要：科学技術シミュレーションの分野において、連成計算は今後必要不可欠な実行シナリオとなっている。連成計算の実行シナリオを有効に利用することによって、複数のアプリケーションを多拠点に配置し、より大規模な解析を行うために、数拠点利用を実現する拠点連携システムを提案する。これは、各拠点のスケジューラといった基盤システムの協調と拠点間メッセージの管理を行うための Point-of-Presence サーバと、アプリケーションに提供される拠点間通信 API からなり、全てをユーザ権限で実施可能な Well-known サービスで構成することにより、環境に対して柔軟に、最低限の労力で協調環境を提供可能とする。本研究ではこのシステムの設計と実装を行った。また今後の追加実装に向けて連成アプリケーションの他拠点実行をエミュレーションすることにより基本的な評価を行った。

1. はじめに

科学技術シミュレーションの分野において、大規模計算機資源を利用することによってこれまで達成不可能であった大規模、詳細なシミュレーションが可能になってきた。中でもさらなる大規模化あるいは、より複雑な物理現象を解明すべく、複数のアプリケーションを組み合わせて利用する連成計算は今後必要不可欠な実行シナリオとなっている。

このような連成計算の一種である、マルチスケールアプリケーションは、大規模な範囲を対象とするアルゴリズムと、より詳細な小さな範囲を計算するアルゴリズムを使い分け、計算資源を効率良く利用することにより、詳細で複雑な物理過程のシミュレーションを可能としている (図 1)。また、陸地（剛体）と海域（流体）を同時に加味したような複雑な物理現象を解析するにはそれぞれを対象とする複数のアルゴリズムを組み合わせる必要があり複数のアプリケーションによる連成計算を行うことが多い。

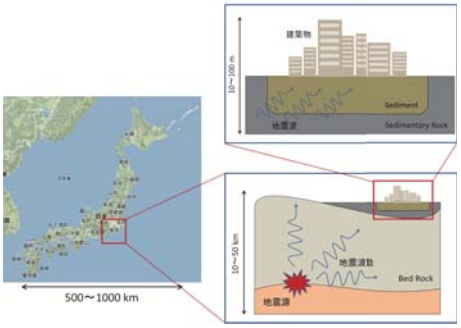

図 1　地震波動シミュレーションにおけるマルチスケール

一方、大規模計算資源を有する拠点の増加に伴い、HPCI[1] のようにこれを協調して利用するインフラが現れている。HPCI には様々な資源が接続されているが、主として、シングルサインオンを実現する認証部分と、データを拠点間で共有する共有ストレージから構築されており、複数拠点を協調してアプリケーションを同時実行する環境とは成っていない。これは、並列プロセスが互いに密な通信を行うような一体型のアプリケーションで複数拠点を利用する場合、レイテンシの大きい拠点間通信の頻度がボトルネックとなり、設計時点から最適化を行わない場合大きく性能が

[1] 東京大学情報基盤センター　〒 113-8658 東京都文京区弥生 2-11-16
Information Technology Center, The University of Tokyo 2-11-16 Yayoi, Bunkyo-ku, Tokyo, 113-8658, Japan
[2] 帝京大学福岡医療技術学部　〒 836-8505 福岡県大牟田市岬町 6-22
Faculty of Medical Technology, Teikyo University 6-22 Misaki-machi, Oomuta, Fukuoka, 836-8505, Japan
[3] 理化学研究所計算科学研究機構　〒 650-0047 兵庫県神戸市中央区港島南町 7-1-26
RIKEN Advancced Institute for Computational Science 7-1-26, Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo, 650-0047, Japan
[4] 東京工業大学学術国際情報センター　〒 152-8550 東京都目黒区大岡山 2-12-1
Global Scientific Information and Computing Center, Tokyo Institute of Technology 2-12-1 Ookayama, Meguro-ku, Tokyo, 152-8550, Japan
[a)] jitumoto@gsic.titech.ac.jp
[b)] tkoba@cc.kyushu-u.ac.jp
[c)] matsumoto@cc.u-tokyo.ac.jp
[d)] shinichiro.takizawa@riken.jp
[e)] miura@gsic.titech.ac.jp

低下してしまうためである。しかし、本研究の対象とするマルチスケールアプリケーションでは、密な通信はアプリケーション内に閉じており、連成される複数のアプリケーションをそれぞれの拠点内で実行し、拠点間でアプリケーション間通信行うよう配置することにより大きな設計の見直しを行わなくても比較的効率良く多拠点を用いた実行を行うことができ、より大規模に計算資源を利用できると考える。

アプリケーションを多拠点に配置する場合、現状では、アプリケーション間は中途ファイルをやり取りするが、このサイズが非常に大きなものになることが多い。一例として、地震波動計算の大規模範囲用アプリケーション SEISM3D[3] を挙げると、10km x 20km x 1km という比較的小さな範囲のシミュレーションにおいても 1 ステップ数 MB、標準的に行われる数万ステップになると、数 TB の出力を、他のアプリケーション用に出力することになる。このような大規模ファイルのやり取りは、可視化を伴うシミュレーションでも用いられる。東大・神戸大で共同研究を行っている粒子計算アプリケーション PARMER3D[2] では現在、衛星規模の計算を行っているが、結果の可視化を行うための出力ファイルが数 10TB 規模になっている。単純に、アプリケーションが結果を出力してから結果ファイルを他拠点に転送、他拠点で次のアプリケーションを実行、という手法をとる場合、アーカイブ化、転送に数日という長時間がかかることも多く、効率の良いデータ転送を考える必要がある。

さらに、多拠点を用いたアプリケーションの実行には、スケジューラ等の基盤システムの拠点間連携も欠かせない。ナイーブに考えた場合、複数のアプリケーションを多拠点で協調させるためには、それぞれのアプリケーションが同時期にそれぞれの拠点で実行されている必要があるが、スケジューラへのジョブの投入から実行開始までには予測の難しいタイムラグが存在するため実現が難しい。しかしながら、統一的なメタスケジューラを構築するには各拠点の運用ポリシのすりあわせや、基盤ソフトウェアのカスタマイズといった大きなコストを必要とする場合が多い。

これを解決するために、本研究では多拠点を利用した連成計算アプリケーションを構成・実行するための導入障壁の低いフレームワークの構築をおこなう。このフレームワークは、拠点の基盤システムをユーザレベルで連携させる機構とアプリケーション間の通信を行うジョブ間連携通信 API からなり、多拠点対応の細かな調整をユーザが行うことにより、各拠点のシステム改装コストや、ポリシ調整の折衝といったコスト発生を防ぎ、なるべく平易に素早い協調環境を作成することを可能とする。また、対象アプリケーションを連成計算アプリケーションに限定することにより、効率よく各拠点の基盤システムの協調、および拠点間データの転送を行うことを目指す。

2. 関連研究

拠点間連携を行う既存手法として、グリッド・コンピューティングがあげられる。これは、それぞれ異なった運用ポリシを持つ計算リソースを統合的に管理し、計算リソースを必要としているユーザに適切に供給する手法である。NAREGI[4] を始め、グリッド・コンピューティングを実現するミドルウェアは複数存在するが、シングルサインオン、資源の一元的管理、必要とされる計算資源の検索、MPI 等の従来ミドルウェアの多拠点対応等の要素を兼ね備えた複雑なシステムであることが多い。これらのツールは、各拠点の運用ポリシに適合させるためのカスタマイズを必要とする場合も多く、設置に大きなコストが必要となる。

このような問題を解決するために、グリッド・コンピューティングに求められる機能のうち、必要なサブセットのみを実装することにより拠点に求められる要件を簡素化、導入障壁を下げる研究も存在する。研究コミュニティ形成のための資源連携技術に関する研究 (RENKEI) で用いられている RENKEI-PoP/RENKEI-VPE[5] はこのような戦略をとったシステムである。RENKEI-PoP は、拠点それぞれの持つ共有ストレージを単一に統合するのではなく、A) 拠点それぞれの共有ストレージとは別に、B) 拠点間で共有可能なグローバルストレージを作成し、A),B) 双方をマウントした Point of Presence(PoP) サーバを用いることで緩やかなデータ共有を実現する物である。また、RENKEI-VPE は VPN 接続/VM といった仮想化技術を利用することにより、複数拠点のノードを仮想的に同一セグメントに配置し、複数拠点を用いたアプリケーションの実行を支援する。これらの技術は HPCI の構成手法にも影響を与えている。しかしながら、各拠点において、運用中の計算機に新たなサービスを導入するには、保守管理や安全性の検証といった面から大きな障壁がある。前述した RENKEI-VPE は本研究が目的とする広域分散した計算資源を活用するフレームワークを実現しているが、いずれも管理者レベルの権限を必要とし、拠点間の合意が欠かせない。この問題を解決するために、本研究ではユーザ権限と ssh といった一般的に利用可能と考えられるログインサービスのみで、連成計算が行える様な仕組みを目指しており、その点で有意である。

3. ユーザ駆動型・拠点協調フレームワーク

3.1 想定環境

3.1.1 拠点構成

「京」コンピュータを始め、多くのスーパーコンピュータは、数台のログインノード、実際の計算を行う多数の計算ノードを持つ。これに対し、ユーザはログインノードを通じてスケジューラにジョブを定義したジョブスクリプト

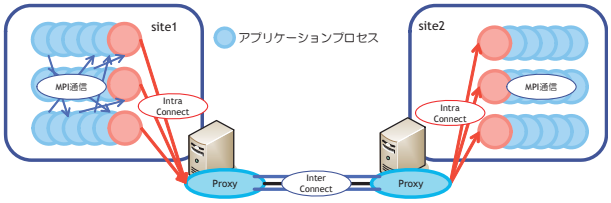

図 2　想定するアプリケーションの実行形態

を投入、スケジューラはこれに従い、各計算ノードにアプリケーションプロセスを配置する。一般的に、ユーザは外部からログインノードを利用することは可能であるが、直接計算ノードに接続することはできない。このため、アプリケーションの協調にはログインノードをプロキシとして利用する必要がある。本研究では以下を拠点の構成として想定している。

- ログインノードが存在し、ssh にてログインすることが可能である
- 計算ノードには直接ログインできず、スケジューラがプロセスを配置する。
- ジョブ実行中に共有ストレージ、もしくは内部ネットワークでの ssh を通じてログインノードと計算ノード間で可能である。
- ログインノード上でユーザプロセスを一定時間実行することが許されている

3.1.2　連成アプリケーション

主として時間発展型のアプリケーションでかつ、頻度の低い連成通信を複数のグループ間で行う連成アプリケーションを対象とする。このようなアプリケーションではある時間単位毎に計算結果をファイルとして出力し、後に可視化や他の解析を行う。連成方法としては、後述するジョブスケジューリングの問題によりアプリケーション間に一方向通信のみしかないことを仮定して開発を行い、双方向通信に関しては今後の課題とする。想定する拠点構成と合わせた場合、この連成アプリケーションの実行形態は図 2 のようになる。この実行形態には 3 種類の通信が存在し、1) アプリケーションを並列化したときに発生する 1 サイト内で行われる通信、2) アプリケーション協調のためにいくつかのプロセスからログインノードへの通信、3) ログインノードをプロキシとしてサイト間で協調データをやり取りする通信である。以降 2),3) に関して、それぞれ Inter-Connect, Intra-Connect とする。また、1) に関しては多くの場合 MPI などが用いられるが、アプリケーション内部で完結するものとして、本研究では対象としない。

3.2　設計

本研究で提案するユーザ駆動型・拠点協調フレームワークは、連成アプリケーションを多拠点実行する際に問題となるものとして、1) アプリケーション間メッセージをどのように最適に送受信するか、2) 各拠点の基盤システムの差異を埋めながらアプリケーションを多拠点に同時投入しているように見せかけるにはどうすれば良いかに注目してフレームワークを設計する。また、全ての要素に渡り、可能な限り Well-known なサービスを利用し、想定する拠点環境に大きな変更を加えずに利用できることを目指す。

3.2.1　アプリケーション間メッセージ送受信

第 1 章で述べたとおり、アプリケーション間のデータをファイル形式でナイーブに拠点間移動すると、数日規模の時間がかかってしまう。しかしながら、時間発展型のアプリケーションでは、解析結果は単位時間毎に出力され、全計算が終了せずとも中途結果を得ることが可能である。このため、複数のアプリケーションをパイプラインにつなげることでオンザフライ転送を行い、通信と計算を同時に行うことが可能である。　この際、通信時間と計算時間の釣り合いに関してはアプリケーション毎に異なるため、特に通信時間が計算時間を超過してしまう場合、プロキシとして用いているログインノードでのバッファリングを行うことや、計算を中断し通信の終了を待つといった複数の戦略をとることができる。これに関しては他拠点に存在するアプリケーションの性質や実行状態、ユーザのポリシに応じて、取り得る最適な手法は異なる。このため、連成アプリケーションに知見のあるユーザに戦略を提示し、選択できる形にすることが望ましいと考える。

また、アプリケーション間メッセージ送受信には前述の通り、Intra-Connect と Inter-Connect を用いた通信が必要となる。Inter-Connect に関しては、ログインノードの性質上 ssh を用いたメッセージのトンネリングを行うことで情報の転送が可能である。また、連携拠点は大規模スーパーコンピュータを保有する情報センターと考えられるため数拠点から数十拠点の規模で収まると考えられ、スケーラビリティに関しては耐性が高い。　一方、Intra-Connection に関しては、計算ノードとログインノードの情報共有手法の差異により、複数の手段を用意する必要がある。想定環境は、多くの拠点で備わっている共有ストレージを用いた手法と、ログインノード計算ノード間を繋ぐ管理ネットワークを用いた手法のいずれかを仮定している。しかしながら、計算ノードからログインノードへの ssh が不可能な環境も多く、共有ストレージについてもジョブスクリプトの投入に合わせたステージングを用いている場合、アプリケーション実行中の通信データ更新が難しくなる。また、この実装に関しては共有ストレージの並列入出力機能を最大限に利用するなど、他のシステム研究の成果を考慮し、スケーラビリティ、拡張性の高いものにしておく必要がある。これは、結果ファイルを出力する代表プロセスを利用する時に発生するデータの集約と同期がボトルネックとなってしまうことを避けるために、多くのアプリケーションの実装が複数のプロセスがそれぞれデー

タを出力する形式になると考えたためである。

3.2.2 多拠点ジョブ投入

連成アプリケーションが多拠点で動作するには、単純に設計した場合、各拠点においてアプリケーションが同時に起動している必要がある。しかし、各拠点にジョブスクリプトを同時に投入したとしても、実行キューの待ち時間は想定が難しく、それらが同時に実行状態になる保証はない。また、拠点それぞれが定めるポリシに従い、アプリケーションの実行時間には最大値が決まっており、他拠点で実行されるアプリケーションを無限大に待ち続けることは不可能である。これを解決するために、本研究では 2 種類の手法 1) ログインノードでの通信バッファリングを用いた連成アプリケーションの分解実行、2) ジョブスクリプトの投機投入による後続アプリケーション実行の加速を考える。

3.2.2.1 通信バッファリング

本研究では連成するアプリケーション間の通信を一方向と仮定している。これは、後続アプリケーションが実行される前に、先行アプリケーションの結果データを他拠点に前もって送信する為である。データをパイプライン状に、後続アプリケーションが実行される可能性のある拠点に送信し、バッファリングあるいはファイルとして保存しておくことにより、データを生成するアプリケーションは一切の待ち時間を必要とせず計算を続行することができ、さらに自身の実行時間とデータの送信時間を重ね合わせることが可能になる。後続アプリケーションは起動後、まずバッファリングされたデータを用いて計算を行うことで、あたかも先行プロセスが同時に実行されているように計算をすることが可能である。

3.2.2.2 ジョブスクリプトの投機投入

アプリケーションが特定の拠点に依存しない場合、複数の拠点に同じアプリケーションを投入し、最も早く実行される拠点のみを利用することで、後続アプリケーションの実行を加速することが考えられる。採用されなかった拠点に関しては、実行をキャンセルすることにより、無駄な計算資源の利用も避けることが可能である。

3.2.3 実行環境の維持

想定している実行環境では、アプリケーション間通信をリダイレクションするサーバプロセスがログインノードに配置される。しかしながら、ログインノードでのアプリケーション実行時間は拠点の運用ポリシにより短く限られることが多い。このため、環境が構築された後に、互いを監視し合いいずれかの PoP サーバが終了させられている状況でその拠点への通信が発生した場合再起動を行う必要がある。PoP サーバの終了は外部からの強制終了によって行われるため、データ送信元とデータ送信先での通信の一貫性を保つような耐故障手法を導入する必要がある。これに対応するため、拠点間通信ではデータ受信側でバッファリングされたメッセージに関して ACK をローカルに保存すると同時に生成側に返信し、生成側で ACK を受信次第同様にローカルに保存しておくという手法を用いる。データ生成側と受信側でそれぞれ保存された ACK データを比較することで、通信の一貫性を取ることができる。

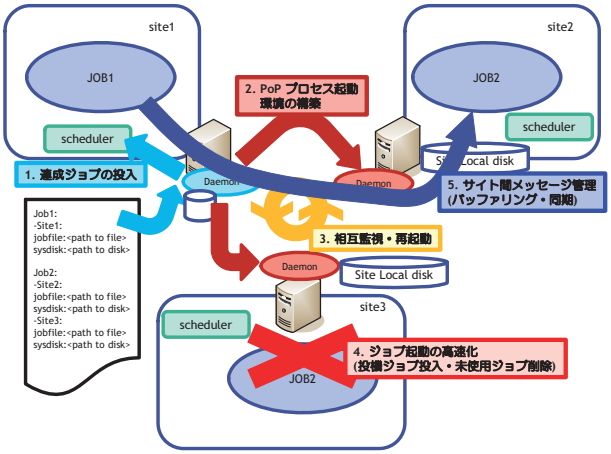

図 3 システムの動作シナリオ

3.3 実装

3.2 節で提示した設計を元に、以下のような実装を行っている。実装は大きく分けて 1) ログインノードをプロキシとして使うための PoP サーバ、2) アプリケーションが拠点間通信を行うための P2P 通信 API の 2 つであり、動作シナリオを含むシステムの全体図は図 3 のようになる。

また、設計で挙げた項目のうち以下は簡易性の為に後の課題とした。

- Intra-Connection における ssh による実装
- ジョブスクリプトの投機投入
- 実行環境の維持

3.3.1 PoP サーバ

PoP サーバは各拠点の基盤システムの連携、およびメッセージのリダイレクションを行うアプリケーションであり、次のように動作する (図 3)。

(1) ユーザの連成アプリケーション投入に応じて、他拠点に PoP サーバを実行する。この際、ssh の機能であるトンネリングを用いる。この際、後に行う PoP サーバの相互監視・再起動の為に、各拠点へログインするためのパスフレーズ（非推奨だがパスワードも可）をユーザに問い合わせ保持しておく。

(2) ユーザの連成アプリケーション投入の内容に従い、先行アプリケーションをスケジューラに投入する。

(3) 先行アプリケーションの実行開始後、もしくは、先行アプリケーション投入からユーザが指定する時間の後に後続アプリケーションを他拠点にて投入する。この際ユーザの指定があれば投機投入を行う。

(4) 後続アプリケーションのいずれかが実行開始された時

に、他の投機投入に関してはキャンセルする。

(5) ユーザの指定するポリシに従い、アプリケーション間のメッセージを集約、拠点間で送受信する。全体のメッセージの流れについては 3.3.2 節にて説明する。

3.3.2 アプリケーション間通信 API

アプリケーション間通信 API は送信プロセスと受信プロセスを識別するタグを設定可能なブロッキング P2P 通信 API(Send/Receive) である。同じプロセスから送出されたメッセージに関してのみ順序が保存される。識別タグに関しては、送信プロセス内、受信プロセス内でそれぞれ重複のない整数値をアプリケーションプログラマが任意に利用する。多くの場合、連成されるアプリケーションは MPI によって実装されていると考えられるため、それぞれにおけるランクを利用すると重複がない状態が実現できる。　前述の通り、本研究では Intra-Connection に共有ストレージを利用する手法を優先的に実装している。アプリケーションが生成するメッセージは以下のように扱われる。

- 送信 API により、プロセス毎に作成されるファイルに通信内容が記述される。このとき、ユーザが設定ファイルに記載したブロックサイズ毎にデータが分割され、ヘッダを追加される。ヘッダには送信プロセス、受信プロセスの識別タグと、ブロック開始時点での残りデータサイズが記載されている。
- PoP サーバは POSIX 非同期 IO を用いてプロセス毎の通信用ファイルを監視している。ブロックサイズ分のデータを通信用ファイルから読み込めた段階で、SSH トンネリングされた送付先拠点へのソケットへブロッキング送信を行う
- 送付先拠点の PoP サーバは受け取ったデータについてヘッダを参照しながら識別タグのペア毎に作成される通信ファイルに書き込む。
- 受信 API により、受信側のアプリケーションは通信ファイルからヘッダを取り除きつつデータを読み込む。この際、通信ファイルは API を呼び出す際に指定された識別タグによって一意に決定される。

3.3.3 利用シナリオ

ユーザは本システムを以下のように利用する。

(1) 複数のアプリケーションを連成させるために、それぞれのアプリケーションを拠点間 P2P 通信 API を用いて改変する。これは、連成するアプリケーションの作成者が協調して行われることが望ましい。

(2) 連成アプリケーションの投入スクリプトを作成する。ここには、連携させる拠点のログインノード名や各拠点で実行するジョブ投入スクリプトファイル名、アプリケーションの依存関係を記述する。

(3) 拠点毎で実行されるジョブ投入スクリプトを用意する

(4) 依存関係に従って最初に実行するアプリケーションを配置する拠点で作成した連成アプリケーションの投入スクリプトと共に PoP サーバを起動する。

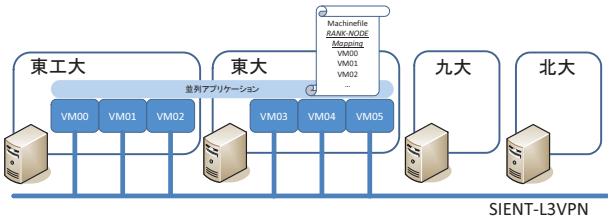

図 4　先端ソフトウェア運用基盤を用いた連成アプリケーション多拠点実行

4. 連成アプリケーションの多拠点実行評価

ここまで説明してきたフレームワークと並行し、多拠点を利用した連成アプリケーションの有効な利用の為の基礎評価を行った。実装が未完成のため、HPCI 先端ソフトウェア運用基盤を用いた多拠点環境のエミュレーションにより評価を実現した。

4.1 HPCI 先端ソフトウェア運用基盤 (HPCI-AE)

先端ソフトウェア運用基盤 (HPCI-AE) は、関連研究でもある RENKEI-PoP/RENKEI-VPE[5] を用い、複数拠点に存在する VM ホスティングリソースに対して、VM の実行管理を利用者主体で行えるシステムである。これは HPCI の枠組みの中で提供されており、北海道大学、東京大学、東京工業大学、九州大学が資源を提供している。これを利用することにより、複数拠点の計算リソースを一つのネットワークセグメントに配置することが可能で有り、多拠点に対応していない通常の連成アプリケーションにおいてもプロセスの配置を工夫することで、簡易に多拠点実行の評価を行うことができる。(図 4)。

4.2 FDM ＋可視化アプリケーションの多拠点利用

連成計算アプリケーションとして、3 次元熱伝導方程式の有限差分法（FDM）解析コードに可視化処理用のルーチンを導入したもの用いた。このコードは FDM 解析部と可視化処理部がそれぞれ特定の MPI プロセスで分割されており、HPCI-AE でのエミュレーションに向いた構造である。

4.2.1 評価環境

VM ホストとして東京工業大学に 4core/1 ホスト（ホスト A)、九州大学に 1core/1 ホスト（ホスト B）を用意した。それぞれのホストは 1core 当たり 3GB である。FDM 解析部では、熱伝導方程式の空間微分を 2 次精度中心差分で評価し、1 次精度オイラー陽解法で時間積分を行っており、OpenMP により並列化されている。全格子点数は 64 × 64 × 64、時間発展のためのメインループの iteration 回数は 1000 とした。可視化処理部は FDM 解析部から全格子点数分の温度データの受信をノンブロッキング通信で行

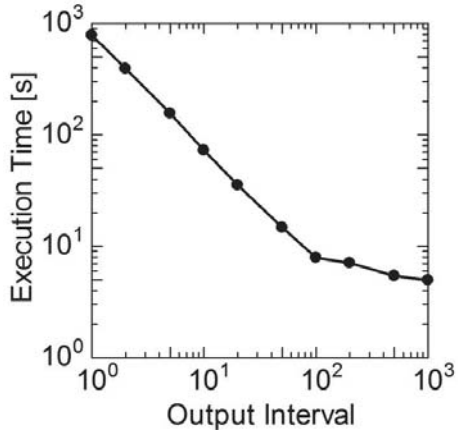

図 5　データ出力間隔に対する実行時間

い処理を行う。

4.2.2　最適な通信頻度と多拠点利用の有用性

多拠点利用の有用性を示す評価として、1) ホスト A のみで FDM 解析と可視化を行う場合と、2) ホスト A,B 双方に FDM 解析と可視化を分散して行う場合の実行時間を測定する。FDM 解析部は 4 並列、可視化部は 1 並列で実行される。　評価に先立ち、環境に合わせた適切な拠点間通信頻度を測定するため、2) の条件を用い、FDM 解析結果の出力間隔に対する実行時間を測定した。これを両対数グラフで示したのが図 5 である。ここでデータ出力間隔とは、「FDM 解析部メインループの iteration 中で何回に 1 回、可視化処理部を呼び出すか」を示す値であり、例えばデータ出力間隔＝ 10 であれば、FDM 解析ループ 10 回に 1 回だけ可視化処理部を呼び出し、データを出力する、ということを表す。同図より、データ出力間隔＝ 100 を境に実行時間が大きく異なることがわかる。データ出力間隔を 100 以下に設定すると、FDM 解析部の実行時間に対して可視化処理部の実行（MPI 通信）に時間がかかることで、全実行時間が大幅に増加する一方で、データ出力間隔を 100 以上に設定すると、可視化処理部の実行が FDM 解析部の実行時間に隠ぺいされる傾向にあることから、実行時間がほぼ一定となる。

以上の測定結果より、実行時間がほぼ定常となる出力間隔 100 を設定した上で、1), 2) の測定を行った。上記の検証と同じく、並列化数は、FDM 解析部 4 並列、可視化処理部 1 並列のため、1) ではサイトの能力以上の並列処理が走るようになる。この結果、1) では 9 sec.、2) では 8 sec. の実行時間となった。この結果は適切なパラメータ設定が行われた場合、多拠点連携処理に効果がある可能性が示唆される。

5.　おわりに

多拠点を利用した連成計算アプリケーションを構成・実行するための導入障壁の低いフレームワークの提案を行った。これには、ssh に代表される Well-known サービスを用い、各拠点の計算資源運用ポリシに配慮した上で、これまでのグリッド・コンピューティングに代表されるような煩雑で時間的コストの大きい拠点間交渉を行わずに、連成計算環境を構築することを目指したものである。　また、実際に多拠点を用いた連成アプリケーション実行が単拠点実行と比較して有意かを検証し、適切なパラメータ設定において優位なケースが存在することを確認した。　今後の展開としては、実装の完成とともにより大規模での多拠点利用検証を行うことを考えている。対象としては ppOpen-HPC[8] ライブラリ群を利用した地震波動－建築物振動連成解析アプリケーションと可視化プログラムの組み合わせを考えている。この連成アプリケーションは最大の問題サイズとして、東大 FX10 システム 4560 ノード（FDM: 2560 ノード＋ FEM: 2000 ノードの同時実行）を使った大規模連成解析の実績があるものである。　また、フレームワーク実装としては、今回の評価時に前実験として欠かせない要素であった適切な拠点間通信量の策定を自動化する枠組みを加えることや、大規模解析によって得られた最適化に関する知見を導入し完成度を高めるコトを考えている。くわえて、大規模解析に利用する前述したアプリケーションは、FDM+FEM+可視化の 3 アプリ連成になっており、可能であれば多アプリ連成が可能なフレームワークへの拡張を行いたいと考えている。

謝辞　本研究の一部は，学際大規模情報基盤共同利用・共同研究拠点 (JHPCN) の支援による．

参考文献

[1] High Performance Computing Infrastructure, https://www.hpci-office.jp/
[2] Hideyuki Usui, Masanori Nunami, Toseo Moritaka, Tatsuki Matsui and Yohei Yagi, A Multi-Scale Electromagnetic Particle Code with Adaptive Mesh Refinement and Its Parallelization, Procedia Computer Science, Vol.4, pp.2337-2343, 2011
[3] Futoshi Mori, Masaharu Matsumoto and Takashi Furumura: Performance of the FDM Simulation of Seismic Wave Propagation using the ppOpen-APPL/FDM Library on Intel Xeon Phi Coprocessor, 11th International Meeting High Performance Computing for Computational Science (VECPAR2014), Eugene, Oregon, USA, June 2014
[4] NAtional REsearch Grid Initiative, https://www.naregi.org/ca/index.html
[5] Shinichiro Takizawa, Satoshi Matsuoka, Masanaru Munetomo, Taizo Kobayashi and Hideyuki Jitsumoto, A Virtual Machine Hosting System on e-Science Cyber infrastructure, The 1st International Workshop on Cloud Computing and Applications (IWCCA 2012), Dec. 2012.
[6] Kenjiro Taura, GXP : An Interactive Shell for the Grid Environment, Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA'04), 2004
[7] Yoshikazu Kamoshida and Kenjiro Taura, Scalable Data Gathering for Real-time Monitoring Systems on Distributed Computing, Cluster Computing and the Grid

(CCGrid), 2008 8th IEEE International Symposium on, May. 2008.
[8] Nakajima, K., Satoh, M., Furumura, T., Okuda, H., Iwashita, T., Sakaguchi, H., and Katagiri, T., ppOpen-HPC: Open Source Infrastructure for Development and Execution of Large-Scale Scientific Applications with Automatic Tuning (AT), Proceedings of PASC 14 (The Platform for Advanced Scientific Computing), Zurich, Switzerland, 2014
[9] 實本英之, 小林泰三, 松本正晴, 滝澤真一朗, 三浦信一, 中島研吾, 複数拠点利用を実現するユーザ駆動型・拠点協調フレームワーク, 電子情報通信学会技術報告, 2014 年 7 月