グリッド上のアプリケーション連携のための枠組みの提案とそのシステムの構築

下坂久司[†] 廣安知之[††] 三木光範[††]

近年，グリッド技術は Web サービスを基盤としたサービス連携に主眼をおいて標準化が図られている．本論文ではアプリケーション連携に特に注目し，グリッド上のアプリケーション連携の枠組みを提案する．提案する枠組みは Notification を用いたアプリケーション実行の連鎖を基盤としており，アプリケーション所有者は提案する枠組みに従い，既存アプリケーションを Web サービスとして表現することで，エンドユーザによる任意のアプリケーション連携を可能にする．本研究では提案する枠組みに基づくグリッド上のアプリケーション連携システムを Application Igniting System と呼び，有効性を検討するために Globus Toolkit を用いて実装した．提案する枠組みにより，Web サービスとして表現された既存アプリケーションの連携による，効率的な問題解決が期待できる．

Proposal of the framework of application integration on the Grid and system construction

Hisashi Shimosaka,[†] Tomoyuki Hiroyasu[††] and Mitsunori Miki[††]

Recently, Grid technologies have been standardized with the emphasis on Web service integration. This paper especially focuses on application integration. Then, we propose new framework of application integration on the Grid. The proposed framework is based on the notification framework for integrating applications. By application holders expressing existing applications as the Web service in keeping with the framework, an end-user can design arbitrary application integration over the wide area network. In order to discuss the effectiveness of the framework, we implement the application integration system on the Grid, application igniting system, using the Globus Toolkit. Through the discussion, it can be expected that the framework realizes the effective problem solving by integrating existing applications expressed as the Web service.

1. はじめに

自動車や航空機の設計，バイオインフォマティクスなどの複合領域にわたる問題解決では，異なる人や組織から提供される複数のアプリケーションを統合して利用できることが望まれる．このような問題解決を支援する一般的なシステムは，提供されるアプリケーションへの統一的なアクセス方法を用意し，エンドユーザによる任意のアプリケーション連携により問題解決を図る．一方でアプリケーション実行には専用の装置やデータベース，並列計算環境を必要とする場合があり，その他にも計算環境に適したチューニングや，ライセンスによる実行可能な計算資源の制限などから，広域ネットワークを通じてアプリケーション連携を支援するシステム構築が有効であると考えられる．

広域ネットワーク上のシステム構築には，グリッド[1),2)] の利用がシステムの実用性や開発コストを考慮した場合不可欠である．最近では，Global Grid Forum(GGF) において Open Grid Services Architecture(OGSA)[2),3)] が提案されており，Web サービスを基盤技術として，資源間の連携に主眼を置いた標準仕様の策定が進められている．

このような背景から本研究ではアプリケーション連携に特に着目し，Web サービスを基盤としたグリッド上のアプリケーション連携の枠組みを提案する．アプリケーション所有者は提案する枠組みに従い，既存アプリケーションを Web サービスとして表現することで，エンドユーザによる広域ネットワークを通じた任意のアプリケーション連携を可能とする．提案する

† 同志社大学大学院
Graduate School of Engineering, Doshisha University
†† 同志社大学工学部
Department of Knowledge Engineering, Doshisha University

枠組みは Notification[4] によるアプリケーション実行の連鎖を基盤としており，アプリケーション実行を逐次に連鎖させるだけでなく，複数のアプリケーションを並列に実行したり，アプリケーション連携を別のアプリケーションから呼び出して利用できる仕組みもサポートとする．また提案する枠組みに基づいたグリッド上のアプリケーション連携システムを，Application Igniting System と呼ぶ．本研究では提案する枠組みの有効性を検討するために，ADVENTURE プロジェクト[5] により提供される複数のアプリケーションを用いて，構造解析および構造最適設計に適用した．

2. Web サービスと Notification

近年，広域ネットワーク上に分散して配置された資源を，仮想的に統合して利用するための基盤技術であるグリッドが高い注目を集めている．最近では GGF において，ビジネス分野におけるグリッドの利用促進や，グリッドミドルウェア間の相互運用性の確保などを目的として，OGSA が提案されている．OGSA は状態管理技術を有する Web サービスを基盤技術とし，標準化されたメタ OS サービス群を定義する．状態管理技術を有する Web サービスにおいて，状態変化を扱う操作は非常に重要であり，その 1 つに Notification がある．

Notification は Web サービスの状態変化により駆動する，登録型の非同期メッセージ通知の枠組みである．Notification にはいくつかの仕様が存在するが，最も単純な Notification の仕組みを**図 1** に示す．図 1 においてサービス A の状態変更通知を受けたいエンドユーザやサービス B は，あらかじめサービス A に対してそれぞれ通知予約 (subscribe) を行う．その後，なんらかの処理によりサービス A の状態に変更が加えられた際には，通知予約のあったエンドユーザおよびサービス B に対して，サービス A から状態の詳細が記述されたメッセージとともに，状態変更が通知 (notify) される．これによりサービス A の状態変化に基づいて，エンドユーザやサービス B において別の処理を連鎖的に実行することが可能となる．

3. グリッド上のアプリケーション連携

本研究では，Web サービスを基盤としたグリッド上のアプリケーション連携の枠組みを提案する．また提案する枠組みに基づいたグリッド上のアプリケーション連携システムを Application Igniting System と呼ぶ．本章では提案するアプリケーション連携の枠組みについて述べ，その後，Application Igniting System について述べる．最後に，提案する枠組みを構造解析および構造最適設計の問題解決に適用する．

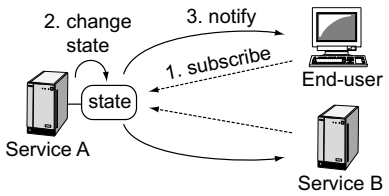

図 1 Notification の概要
Fig. 1 Overview of the notification framework

3.1 アプリケーション連携の必要要件

一般にアプリケーション実行は，複数ファイルの入出力に置き換えることができる．アプリケーションのソースコードの変更を必要最小限とすることを考慮すると，アプリケーション間で入出力ファイルを交換することにより連携を行う方法が最も現実的である．一般的なアプリケーション連携システムでは，エンドユーザはまず各アプリケーションの概要，入出力ファイルの数や書式などの情報をシステムから取得する．その後，アプリケーションの実行順序と，エンドユーザを含めたアプリケーション間におけるデータ交換を定義することにより，アプリケーション連携を設計する．データ交換の定義では，あるアプリケーションの出力ファイルを別のアプリケーションの入力ファイルとするだけでなく，交換される入出力ファイルの書式が異なる場合には，書式の変換方法も併せて指定する．

一方で最適化計算などの複雑な問題解決においては，エンドユーザが設計したアプリケーション連携を，別のアプリケーション実行中に呼び出して利用できる仕組みが必要となる．構造最適設計を例にとると，構造解析を行うアプリケーション連携を，汎用最適化アプリケーションから繰り返し呼び出し，探索点に対応する解析結果を得ることで，ある環境における最適構造を決定することが可能となる．

また負荷分散を考慮した場合，集中管理によってアプリケーション実行やデータ交換を制御するのではなく，サービスとして表現されたアプリケーション間の分散管理が望ましい．本研究では 2 章で述べた Notification の利点を踏まえ，各サービスの分散管理によるアプリケーション連携の枠組みを提案する．

3.2 アプリケーションの実行順序

提案するアプリケーション連携の枠組みにおけるアプリケーション実行順序は，Notification によるアプリケーション実行の連鎖により決定される．ここでアプリケーション連携全体の実行時間の短縮を考慮した場合，逐次的にアプリケーション実行を連鎖させるだ

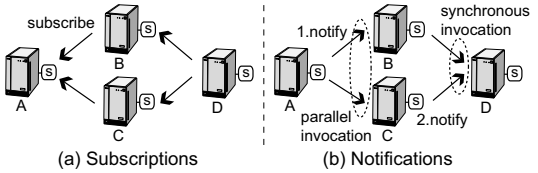

図 2 アプリケーション実行の連鎖
Fig. 2 Chain of application invocations

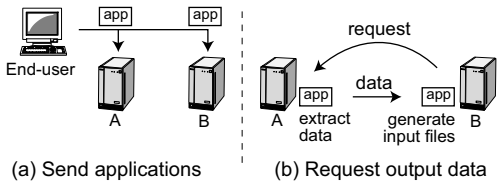

図 3 書式の変換
Fig. 3 Translation of file format

けでなく，アプリケーションの並列実行および同期実行をサポートする必要がある．このようなアプリケーション連携を実現するために，アプリケーションを表現する各 Web サービスは，アプリケーション実行後に更新される「状態 S」と，状態変更通知によりアプリケーションを実行する機能を共通して保持する．アプリケーションを実行する機能は，複数の状態に通知予約を行っている場合，全てのサービスからの状態変更通知を待ち，同期をとってアプリケーション実行を開始する．

提案する枠組みにおいてエンドユーザは，実行したいアプリケーション実行順序の逆順にサービス間の通知予約を構成することで，アプリケーション連携を設計する．ここで，**図 2**(a) に示すようにサービス間の通知予約を構成した場合，サービス A のアプリケーション実行後に状態 S が更新され，図 2(b) に示すようにサービス B，C に状態変更が通知される．これにより両サービスにおいて並列にアプリケーション実行が開始される．またサービス D ではサービス B，C に通知予約を行った情報を保持し，両サービスからの状態変更通知を待ち，同期をとってアプリケーション実行が開始される．Notification をアプリケーション実行順序の決定に用いる利点を以下に示す．

- エンドユーザがアプリケーション実行順序を任意に決定できる．
- 複数サービスへのメッセージ通知により，アプリケーションの並列実行を実現できる．
- 通知予約先の情報を保持することで，アプリケーションの同期実行を実現できる．

3.3 アプリケーション間のデータ交換

アプリケーション間のデータ交換は，交換される入出力ファイルをエンドユーザが指定する方法，書式の変換方法をエンドユーザが定義する 2 つによって実現される．交換される入出力ファイルの書式が同一の場合には前者が，異なる場合には後者が用いられる．アプリケーション間のデータ交換は，エンドユーザの指示に従い，各サービスにおいてアプリケーション実行前に行われる．これにより複数サービスにおいてアプリケーション実行が並列に行われる際も，アプリケーション実行前に各サービスにおいてデータ交換を並列に実現できる．

3.3.1 交換される入出力ファイルの指定

あるサービス A の出力ファイルをそのままサービス B の入力ファイルとする場合，エンドユーザはあらかじめサービス B に対し，サービス A の出力ファイルを取得するよう指示する．これによりサービス B では，状態変更通知を受け取ってアプリケーション実行を開始する前に，出力ファイルの取得を行うことができる．

3.3.2 書式変換方法の定義

交換される入出力ファイルの書式の変換は，出力ファイルから必要なデータの抽出と，抽出されたデータを用いた入力ファイルの生成により行われる．エンドユーザは，あらかじめ各サービスの入出力ファイルの書式を参照し，データ抽出用と入力ファイル生成用の 2 つのアプリケーションを用意する．書式を変換してサービス A，B 間で入出力ファイルを交換する場合，エンドユーザは**図 3**(a) に示すように，サービス A に対してデータ抽出用，サービス B に対して入力ファイル生成用のアプリケーションをそれぞれ配布する．さらにサービス B に対して，サービス A からデータを取得するよう指示する．これによりサービス B ではアプリケーション実行前に，図 3(b) に示すように，サービス A に対してデータ要求を行い，サービス A におけるデータ抽出，抽出したデータの返信，サービス B における入力ファイル生成の 3 つの手順により，書式の変換が実現される．

3.4 アプリケーション連携の呼び出し

エンドユーザが設計したアプリケーション連携を，別のアプリケーション実行中に呼び出して利用できる枠組みは，最適化計算などの複雑な問題解決には不可欠である．提案する枠組みにおけるアプリケーション連携の呼び出しは，アプリケーション実行順序の決定と同様に Notification を用いて実現する．またアプリケーション間のデータ交換においても同様の仕組みを用い，異なるアプリケーション連携を複数，並列に呼

び出すことを可能にする．

提案する枠組みにおいてアプリケーションを表現する各 Web サービスでは，新たにアプリケーション連携を呼び出す際に更新される「状態 T」と，他のサービスの状態変更通知によりアプリケーション実行を再開する機能を共通して保持する．あるアプリケーションにおいてエンドユーザが設計したアプリケーション連携を呼び出す際には，アプリケーション連携の入力ファイルを書き出し，状態 T を更新する．更新後は他のサービスからの状態変更通知を待ち，通知を受け取った後にアプリケーション連携結果を取得して，アプリケーション実行を再開する．ここで複数の状態に通知予約を行っている場合は，全てのサービスからの状態変更通知を待ち，同期をとってアプリケーション連携結果を取得する．

図 2 のサービス A から D で構成されるアプリケーション連携を，サービス E のアプリケーションから呼び出したい場合，エンドユーザは**図 4**(a) に示すアプリケーション連携を設計する．提案する枠組みにおいてエンドユーザは，アプリケーション連携を構成する最初のサービス A から，サービス E の「状態 T」に対して通知予約を指示する．またサービス E から，アプリケーション連携を構成する最後のサービス D の「状態 S」に対して通知予約を指示する．これにより，サービス E のアプリケーション実行中にアプリケーション連携の入力ファイルが書き出され，状態 T が更新されることにより，図 4(b) に示すようにサービス E からサービス A に対して状態変更が通知される．これはアプリケーション連携の実行が開始されたことを意味する．アプリケーション連携終了時にはサービス D の状態 S が更新されサービス E に状態変更が通知されることにより，サービス E においてアプリケーション連携結果が取得され，それを利用してアプリケーション実行が再開される．

ここでアプリケーション間のデータ交換は 3.3 節で述べた仕組みと同様に，サービス A のアプリケーション実行前にアプリケーション連携の入力ファイルを，サービス E から取得するよう指示することで実現できる．またサービス E では，サービス D からの状態変更通知によりアプリケーション実行を再開する前に，同様の仕組みを用い，エンドユーザの指示に従ってアプリケーション連携結果を取得する．

3.5 Application Igniting System

本研究では，提案する枠組みに基づくグリッド上のアプリケーション連携システムを Application Igniting System と呼ぶ．Application Igniting System の

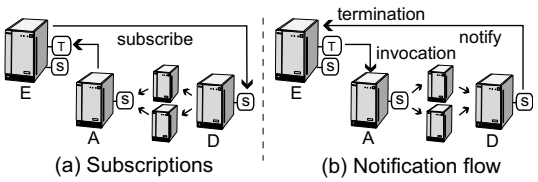

図 4 アプリケーション連携の呼び出し
Fig. 4 Invocation of application integration

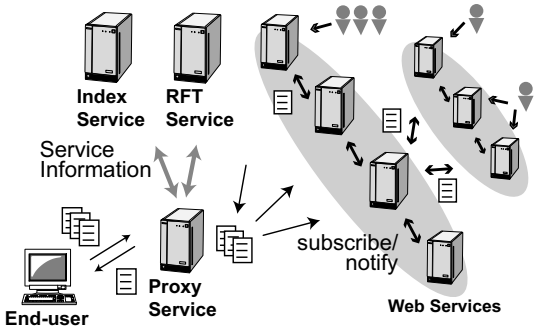

図 5 Application Igniting System の概要
Fig. 5 Overview of the Application Igniting System

概要を**図 5** に示す．本システムでは，アプリケーション連携を補助することを目的とした以下の 3 つのサービスがシステムの実用性を考慮した場合，必要不可欠であると考えられる．例えば RFT サービスを用意しておくことにより，アプリケーション所有者はファイル転送機能を独自に実装することなく，既存アプリケーションを提案する枠組みに沿った Web サービスとして容易に表現することが可能となる．

- Proxy サービス：エンドユーザからの全ての要求，ファイル転送を中継して実行する．プライベートネットワーク上のエンドユーザと，広域ネットワーク上のサービス群との橋渡しを行う．
- Index サービス：サービス群からアプリケーションの概要，入出力ファイルの数，書式などに関する情報を収集し，Proxy サービスを通じてエンドユーザに提供する．
- RFT サービス: 高速で信頼性の高い第 3 者ファイル転送機能を提供する．サービス間のファイル転送は，全てこのサービスの機能を利用して行われる．

3.6 適 用 事 例

本節では構造解析および構造最適設計において一般的に用いられるアプリケーションを対象に，本研究で提案するアプリケーション連携の枠組みを適用する．

3.6.1 構 造 解 析

一般に構造解析は，解析対象の構造を定義する 3D

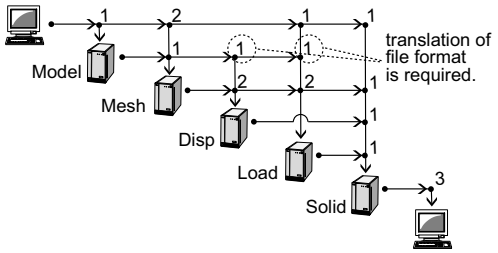

図 6 アプリケーション連携のデータフロー
Fig. 6 Data flow of the application integration

モデリング (Model)，定義した構造のメッシュ分割 (Mesh)，固定する部分を指定する変位境界条件 (Disp)，力の加わる部分を指定する荷重境界条件 (Load)，有限要素法を用いた構造解析 (Solid) の 5 つのステップから構成され，各ステップそれぞれに異なるアプリケーションを適用することが可能である．ここでは仮に，各ステップに対応するアプリケーションが，提案する枠組みに基づき Web サービスとして公開され，**図 6** に示すデータフローがエンドユーザにより設計されたとする．データフローの各交点には，エンドユーザおよび各アプリケーション間で交換される入出力ファイル数が記述されている．ここでエンドユーザは解析の初期設定として，5 つの入力ファイルを 4 つのアプリケーションに提供し，Solid サービスから解析結果として 3 つの出力ファイルを得る．Solid サービスの入力ファイルに注目すると，エンドユーザから提供される 1 つの入力ファイルの他に，Mesh，Disp，Load の各サービスの出力ファイルをそれぞれ 1 つずつ，計 4 つの入力ファイルにより解析を行う．また Model サービスと Disp，Load サービス間のデータ交換は書式の変換を必要とする．図 6 のデータフローを実現するために，エンドユーザは次のようなアプリケーション連携を定義する．

アプリケーションの実行順序： 構造解析におけるアプリケーション実行順序は，各ステップに対応するアプリケーションを逐次に実行する順序で決定できる．一方で図 6 のデータフローに注目すると，Disp および Load サービス間に入出力ファイルの依存関係がない．そのため，アプリケーション連携全体の実行時間の短縮を考慮した場合，Mesh サービスのアプリケーション実行後に，Disp および Load サービスのアプリケーション実行を並列に開始できる．また両アプリケーションの実行終了の同期をとり，Solid サービスにおけるアプリケーション実行を開始することが望ましい．これらからエンドユーザは各サービスの状態 S に対して，**図 7** に示すサービス間の通知予約を構成する．

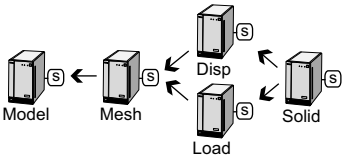

図 7 構造解析のための通知予約
Fig. 7 Subscriptions for structural analysis

データ交換： あらかじめエンドユーザは，4 つのアプリケーションの 5 つの入力ファイルを用意し，各サービスに送信しておく．サービス間のデータ交換は図 6 のデータフローに従い，アプリケーション実行前に取得する出力ファイルを各サービスに指示する．Solid サービスに注目すると，Mesh，Disp，Load の各サービスの出力ファイルをそれぞれ 1 つずつ取得するよう指示する．また Disp および Load サービスはともに，Model サービスの出力ファイルを 1 つ，Mesh サービスの出力ファイルを 2 つ取得するよう指示する．さらに Model サービス間とのデータ交換は書式の変換を必要とするため，Model サービスに対してデータ抽出用，Disp および Load サービスに対して入力ファイル生成用のアプリケーションをそれぞれ配布する．

3.6.2 構造最適設計

一般に構造最適設計は，構造解析を実現するアプリケーション連携を最適化アプリケーションから繰り返し呼び出し，探索点に対応する解析結果を利用することで，ある環境における最適構造を決定する．ここでは 3.6.1 項で述べた構造解析のアプリケーション連携において，最適化アプリケーションにより書き出された探索点情報を Model サービスの入力ファイルとし，Solid サービスの出力ファイルを解析結果として利用するシナリオを想定する．また提案する枠組みに基づき，逐次 2 次計画法に代表される 1 点探索の最適化アプリケーション (SQP) と，遺伝的アルゴリズムに代表される多点探索の最適化アプリケーション (GA) がそれぞれ公開されているものと仮定する．多点探索の最適化アプリケーションでは，探索点情報が複数記述されたアプリケーション連携の入力ファイルを書き出し，構造解析のアプリケーション連携を複数，並列に呼び出す．

アプリケーション連携の呼び出し： アプリケーション連携の呼び出しは，2 種類の通知予約をエンドユーザが指示することにより実現される．1 点探索の最適化アプリケーション (SQP) では，エンドユーザは **図 8**(a) に示す通知予約をサービス間で構成する．ここでエンドユーザは，アプリケーション連携を構成す

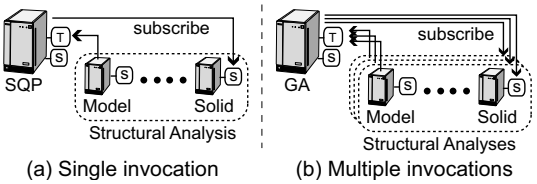

図 8 構造解析アプリケーション連携の呼び出し
Fig. 8 Invocation of application integration for structural analysis

る最初のサービス Model から，SQP サービスの状態 T に対する通知予約を指示する．また SQP サービスから，アプリケーション連携を構成する最後のサービス Solid の状態 S に対する通知予約を指示する．一方で多点探索の最適化アプリケーション (GA) では，アプリケーション連携を構成する各サービスにおいて探索点と同数のインスタンスをあらかじめ生成し，構造解析のアプリケーション連携を複数，個別に設計しておく．その後図 8(b) に示すように，GA サービスと全ての Model サービスのインスタンス間，および全ての Solid サービスのインスタンスと GA サービス間で通知予約を指示する．これにより GA サービスにおける状態 T の 1 度の状態変化により，複数の探索点に対応する個別のアプリケーション連携を，並列に呼び出して実行することが可能となる．

データ交換: アプリケーション連携の呼び出しにおけるデータ交換は，最適化アプリケーションにより書き出された探索点情報を，Model サービスのアプリケーション実行前に入力ファイルとして取得するよう，エンドユーザがあらかじめ指示することで実現する．また最適化アプリケーションに対しては，状態変更通知によりアプリケーション実行を再開する前に，Solid サービスの出力ファイルをアプリケーション連携結果として取得するよう指示する．ここで交換される入出力ファイルの書式が異なる場合は，各サービスにアプリケーションを配布することで書式を変換する．1 点探索の最適化アプリケーション (SQP) では，SQP サービスに対して探索点情報からデータを抽出するアプリケーション，Model サービスに対して入力ファイルを生成するアプリケーションをそれぞれ配布する．同様に，Solid サービスに対して解析結果からデータを抽出するアプリケーション，SQP サービスに対してアプリケーション連携結果を生成するアプリケーションを配布する．一方で多点探索の最適化アプリケーション (GA) では，全ての Model サービスのインスタンスに対して入力ファイルを生成する同一のアプリケーションを配布しておき，GA サービスに対しては，データ要求元となる Model サービスのインスタンス情報から，異なるデータを抽出するアプリケーションを配布する．これにより Model サービスの各インスタンスにおいて，異なる探索点による解析対象構造を定義できる．同様に，Solid サービスの全てのインスタンスに対して解析結果からデータを抽出する同一のアプリケーションを配布しておき，GA サービスに対しては，取得した全てのデータからアプリケーション連携結果を生成するアプリケーションを配布する．これにより複数のアプリケーション連携を並列に呼び出した際にも，エンドユーザの指示に従った書式の変換を実現することが可能となる．

4. 性能評価

本研究では 3.5 節で述べた Application Igniting System を，Globus Toolkit を用いて実装した．また実装したシステム上で，ADVENTURE プロジェクトにより開発された複数のアプリケーションを，提案する枠組みに基づいた Web サービスとして表現し，3.6 節で述べた構造解析および構造最適設計に適用した．構造最適設計では 1 点探索の最適化アプリケーションとして逐次 2 次計画法 (SQP) を用いた．ADVENTURE プロジェクトは東京大学などを中心とする，オープンソースの CAE ソフトウェア開発を目的としたプロジェクトであり，構造物の変形や熱，流体の流れ等の力学解析から可視化，最適設計を可能とする独立したモジュール群を提供している．本章では実装したシステム上の性能評価を通じ，提案する枠組みの有効性を検討する．

4.1 評価環境

本研究における性能評価では，**表 1** に示す計算機を各サービスに割り当て，100Mbps の FastEthernet で接続されたクラスタ環境を用いた．また Application Igniting System においてサービス群の情報を収集する Index サービス，第 3 者ファイル転送機能を提供する RFT サービスは，Globus Toolkit の基本サービスをそのまま用いた．ここで RFT サービスは，サービスごとに異なる場所のものを利用可能であるが，本性能評価においては 1 つの RFT サービスを共有して利用した．構造解析および構造最適設計において対象とした構造物の規模は，メッシュサイズ 600 から 700 程度の比較的小規模なものであり，サービス間で転送されるファイルサイズは最大でも 60KB 以下である．

4.2 構造解析事例における性能評価

3.6.1 項で述べた構造解析のアプリケーション連携において，構造解析の各ステップにおける 10 試行の

表 1 計算機のスペック
Table 1 Spec of each node

CPU	Pentium 4 2.80GHz
Memory	512MB
OS	Linux 2.4.27 Debian GNU/Linux 3.0
Software	Globus: 3.2.1, Java: 1.4.2 gcc: 2.95, glibc: 2.2.5

表 2 構造解析事例におけるオーバーヘッド時間
Table 2 Overhead time of application integration for structural analysis

ステップ	状態変更 通知元	状態変更 通知先	オーバーヘッド 時間 [ms]
Model	Proxy	Mesh	166
Mesh	Model	Disp, Load	2865
Disp	Mesh	Solid	5889
Load	Mesh	Solid	5859
Solid	Disp, Load	Proxy	8532

平均オーバーヘッド時間を**表 2** に示す．ここでは通知元に示すサービスから状態変更通知を受信した時間を起点とし，通知先に示すサービスに状態変更が通知されるまでの間の，アプリケーション実行時間を除く平均時間を示した．また Proxy サービスはエンドユーザの要求を中継するサービスであり，Model サービスへアプリケーション実行のための状態変更を通知し，さらに Solid サービスからの状態変更通知により，アプリケーション連携の終了をエンドユーザに伝える役割を担う．

表 2 より，構造解析の各ステップのオーバーヘッド時間は 166 ミリ秒から 8532 ミリ秒とばらつきが大きいことがわかる．特にオーバーヘッド時間が長い Disp と Solid について，その内訳を以下に示す．まず Disp サービスでは，アプリケーション実行前に Mesh サービスの 2 つの出力ファイルを RFT サービスを通じて取得するが，これに 5319 ミリ秒 (90.3%) のオーバーヘッド時間を必要とする．また Model サービスに書式の変換をリクエストし，抽出されたデータから入力ファイルを生成するが，これは 209 ミリ秒 (3.5%) とオーバーヘッド時間は短い．一方で Solid サービスでは，アプリケーション実行前に Mesh，Disp，Load の異なるサービスから出力ファイルをそれぞれ 1 つずつ RFT サービスを通じて取得し，これに 7692 ミリ秒 (90.2%) のオーバーヘッド時間を要する．これらからオーバーヘッド時間のばらつきは，RFT サービスを通じた出力ファイルの取得に大きく依存しており，全体の約 90%を占める．これは Globus Toolkit の RFT サービスの実装上の問題といえ，これを除くと本研究で実装したシステムのオーバーヘッドは小さく，実用上問題ないと考えられる．また提案する枠組みに基づき，アプリケーションを Web サービスとして公開する場合には，RFT サービスの実装がオーバーヘッドに大きく影響を与えるため，アプリケーション実行時間が短い場合には留意する必要がある．

表 3 アプリケーション連携の呼び出しにおけるオーバーヘッド時間
Table 3 Overhead time for invoking application integration

呼び出し操作	オーバーヘッド 時間 [ms]	割合 [%]
証明書の読み込み	554	6.4
リスナーの起動	4967	56.9
状態変更通知 (Invocation)	655	7.5
状態変更通知 (Termination)	1583	18.1
書式の変換	453	5.2
リスナーの終了	200	2.3

4.3 構造最適設計事例における性能評価

次に 3.6.2 項で述べた構造最適設計のアプリケーション連携における，アプリケーション連携の呼び出し部分のオーバーヘッド時間を示す．本性能評価では，1 点探索の最適化アプリケーション (SQP) から構造解析のアプリケーション連携を計 28 回呼び出した際の，アプリケーション連携実行時間を除く平均呼び出し時間を測定した．その結果，平均オーバーヘッド時間は 8772 ミリ秒であり，その内訳は**表 3** のとおりである．

本研究で提案する枠組みでは Web サービスの細かな内部仕様を規定しておらず，アプリケーション所有者は枠組みに沿った実装を任意に行うことができる．ADVENTURE プロジェクトにより提供される最適化アプリケーションでは，他のアプリケーションの呼び出しを，設定ファイルで指定したコマンド実行により行っている．そのため SQP サービスの実装では，以下の手順に従う実行コマンドを設定ファイルに記述し，グリッド上のアプリケーション連携の呼び出しを実現した．実行コマンドでは，最初に SQP サービスのプロキシ証明書を読み込み，Notification リスナーを立ち上げる．その後，探索点情報が書き出されていることを確認し，SQP サービスに対し状態 T の更新をリクエストする．これによりアプリケーション連携の実行が開始される．アプリケーション連携終了時には，SQP サービスに対して状態変更が通知されるため，その情報を Notification リスナーに転送してもらい，アプリケーション連携結果を取得後，Notification リスナーを終了させる．

表 3 より，これらの手順において最もオーバーヘッド時間を要するのは Notification リスナーの起動で

あり，終了と合わせると約 59.2%の割合を占める．これは Globus Toolkit の実装上の問題であり，最適化アプリケーションと実行コマンド間の情報交換では，Notification に依存しない仕組みが望ましいといえる．その他の手順におけるオーバーヘッドは比較的小さく，実用上問題ないと考えられる．

5. 関 連 研 究

アプリケーション連携に関する研究の多くはワークフローに関する取り組みとして多くなされている．科学技術計算分野では計算リソースや大規模なデータセットの取り扱い，実行されるジョブの多様性などから標準的な枠組みはなく，様々なプロジェクトで多様なワークフローエンジンが開発されている[6]．そのなかでも文献[6] では，代表的な 12 のワークフローエンジンをスケジューラとの連携方法やデータセットの取り扱いにより分類している．提案する枠組みでは，スケジューラとの連携方法を規定していないが，Web サービスにおけるアプリケーション実行をスケジューラに委託するモデルを考えると，Triana[7] と類似した分類に属する．

広域ネットワークを通じ最適化計算システムを構築できる既存システムとして，NEOS Server[8] および FIPER システム[9] が挙げられる．NEOS Server は Condor をベースとしたシステムであり，エンドユーザにより定義された最適化計算をネットワーク上の遊休資源上で実行する．NEOS Server では，複数のアプリケーションを連携させる複雑な最適化問題を定義することは難しい．一方で FIPER システムは，J2EE に準拠した ACS サーバと多数の FIPER ステーションで構成され，利用可能なコンポーネントを任意に組み合わせて複雑なアプリケーション連携を設計できる．しかしながら ACS サーバの集中管理によりステーション上でのジョブ実行やデータ交換を制御しており，スケーラビリティに乏しい．そのため大規模なデータセットをやりとりする際などには，本研究で提案する枠組みが有効であると考えられる．

6. まとめと今後の課題

複合領域にわたる問題解決において異なる人や組織から提供されるアプリケーションを，広域ネットワークを通じ連携して利用できることが望まれる．そのため，本研究ではグリッド上のアプリケーション連携の枠組みを提案した．アプリケーション所有者は提案する枠組みに従い，所有するアプリケーションを共通の状態，インターフェース，機能を有する Web サービスとして表現することで，エンドユーザによる任意のアプリケーション連携を可能にする．提案する枠組みは Notification によるアプリケーション実行の連鎖を基盤としており，広域ネットワーク上のアプリケーションの並列実行や同期実行，データ交換，アプリケーション連携の呼び出しなどをサポートする．本研究では提案する枠組みに基づいたグリッド上のアプリケーション連携システムを Application Igniting System と呼び，ADVENTURE プロジェクトにより開発された複数のアプリケーションを用い，構造解析および構造最適設計に適用した．今後の課題として，分岐やループ処理を含むアプリケーション連携，エラー処理の枠組みを提案することなどが挙げられる．

参 考 文 献

1) Foster,I. and et.al., *The Anatomy of the Grid : Enabling Scalable Virtual Organizations*, International Journal of Supercomputer Applications, 2001.
2) Foster,I. and et.al., *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*, Globus Project, 2002.
3) Foster,I. and et.al., *The Open Grid Services Architecture, Version 1.0*, Global Grid Forum OGSA-WG, GFD-I.030, 2005.
4) Graham,S. and et.al., *Publish-Subscribe Notification for Web services, Version 1.0*, 2004, http://www.oasis-open.org/committees/download.php/6661/WSNpubsub-1-0.pdf
5) ADVENTURE Project: http://adventure.q.t.u-tokyo.ac.jp/
6) Yu,J. and et.al., *A Taxonomy of Scientific Workflow Systems for Grid computing*, Special Issue on Scientific Workflows, SIGMOD Record, ACM Press, 2005.
7) Taylor,I. and et.al., *Resource Management of Triana P2P Services*, Grid Resource Management, Kluwer Academic Press, 2004.
8) Czyzyk,J. and et.al., *The Network-Enabled Optimization System*, MCS-P615-1096, Argonne National Laboratory, Mathematics and Computer Science Division, 1996.
9) Rohl,P.J. and et.al., *A Federated Intelligent Product EnviRonment*, 8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, 2000.