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

2006－HPC－ 107 (19)
2006／8／1

グリッド上のアプリケーション連携システムの評価

下坂 久司[†], 廣安 知之[††], 三木 光範[††]

[†] 同志社大学大学院 [††] 同志社大学工学部

Application Igniting System は，WS-Notification を基盤としたグリッド上のアプリケーション連携システムであり，Web サービスとして表現されたアプリケーション間でメッセージ通知を伝搬させることにより，アプリケーション実行の連鎖を実現する．本システムは，アプリケーション所有者に対して，所有するアプリケーションを容易に Web サービス化できる機能を提供する．またエンドユーザに対しては，複数のユーティリティサービスを提供することで，柔軟なアプリケーション連携の設計を可能にする．一方で Application Igniting System は，未だ小規模なアプリケーション連携への適用しかなされておらず，比較的大規模なアプリケーション連携における性能評価が不可欠である．そのため，本研究では 15 種類のアプリケーションを連携させる構造解析事例に Application Igniting System を適用し，その性能を評価した．その結果，メッセージ通知に関連するオーバーヘッド時間は非常に短いことがわかったが，アプリケーション間のファイル交換に要する実行時間が長く，今後の改善が必要であることが明らかとなった．

Evaluation of the Application Integration System on the Grid

Hisashi SHIMOSAKA[†], Tomoyuki HIROYASU[††] and Mitsunori MIKI[††]

[†] Graduate School of Engineering, Doshisha University
[††] Knowledge Engineering Dept., Doshisha University

The "Application Igniting System", which is an application integration system on the Grid based on the WS-Notification, realizes the chain of application invocation by propagating the message notifications among services. In this system, application holders can easily publish their own applications as the Web services. This system also provides some utility services. Then, end-users can design flexible application integration. On the other hand, this system applied only to small-scaled application integration. Therefore, the performance evaluation in large-scaled application integration is indispensable. In this paper, we apply the Application Igniting System to the structural analysis by integrating 15 applications and evaluate its basic performance. Through the performance evaluation, we concluded that the overhead time related to the message notification is very short. However, the overhead time related to the file transfer is long. Then, the file transfer mechanism will be improved.

1 はじめに

複合領域にわたる問題解決では，分野の異なる複数の科学技術計算用アプリケーションを，広域ネットワークを通じて統合利用できるシステムが望まれる．構造物の設計を例に挙げると，構造の解析アプリケーションや最適化アプリケーション，可視化アプリケーションなどをうまく組み合わせて利用する必要があるが，これらのアプリケーションは別の専門組織で開発されていることが一般的である．また各アプリケーションは必要とする実行環境が異なったり，ライセンスの問題から複数の計算機にインストールできない可能性もある．そのため，アプリケーション所有者が最適な実行環境上でアプリケーションを公開し，公開されたアプリケーションを広域ネットワークを通じて，エンドユーザが容易に統合利用できるシステムが有用であるといえる．

このような背景から，我々は広域ネットワークを通じてアプリケーションの統合利用を支援できる Application Igniting System を提案した[1]．このシステムは，WS-Notification (WSN)[2] を基盤としたグリッド上のアプリケーション連携システムであり，Web サービスとして表現されたアプリケーション間でメッセージ通知を伝搬させることにより，アプリケーション実行の連鎖を実現する．アプリケーション実行の連鎖を，以後アプリケーション連携と呼ぶ．Application Igniting System は，アプリケーション所有者に対して，所有するアプリケーションを容易に Web サービス化できる機能を提供する．また複数のユーティリティサービスを提

供することで，エンドユーザによる柔軟なアプリケーション連携の設計を可能にする．しかしながら，現在はまだ小規模なアプリケーション連携への適用しかなされておらず，より大規模なアプリケーション連携における性能の検証を行なう必要がある．そのため，本研究ではADVENTUREプロジェクト[3]により開発された15種類のアプリケーションを用いて，Application Igniting Systemを構造解析事例に適用することで，その性能を評価した．

2 Application Igniting System

Application Igniting Systemは，WSNを基盤としたグリッド上のアプリケーション連携システムである．本章では，まずWSNの概要について述べ，その後，Application Igniting Systemについて述べる．

2.1 WSRFとWSN

近年，Global Grid Forum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，Open Grid Services Architecture (OGSA)[4]の標準化が進められている．OGSAはWS-Resource Framework (WSRF)[5]とWS-Notification (WSN)の2つの仕様に代表されるWebサービス技術を基盤としている．

WSRFは，Webサービスに状態を有するリソースを導入するための仕様である．またWSNは，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．図1に最も単純なWSNの概要を示す．WSNにおいてメッセージの送り手となるサービスはProducerとなり，あらかじめ通知可能なリソースプロパティをトピックとして公開する．エンドユーザや他のサービスはConsumerとなり，興味のあるリソースプロパティを保持するProducerに対して，それぞれ通知予約 (subscribe)を行う．これにより，リソースプロパティに変更が加えられた際には常に，Consumerに対して状態の詳細が記述されたメッセージが通知 (notify)される．Consumerはメッセージ内容を確認し，内容に応じた処理を連鎖的に実行することができる．WSRFおよびWSNを実装したツールにGlobus Toolkit (Globus)[6]のバージョン4.0があり，Application Igniting Systemの構築に用いた．

2.2 Application Igniting Systemの概要

図2にApplication Igniting Systemの概要を示す．各ユーザは，それぞれ以下のように本システムを利用する．以後，アプリケーションの実行順序をワークフロー，アプリケーション間の入出力ファイル交換をデータフローと呼ぶ．

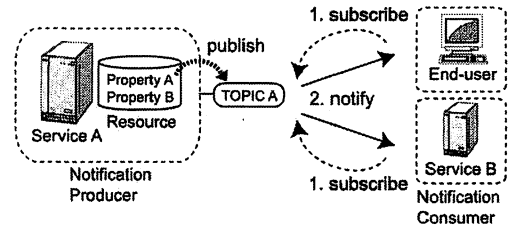

図1: WS-Notificationの概要

- グリッドテストベッド管理者: Globusの提供するIndex (情報収集用)サービスを用意する．また我々の提供するパッケージを用いて，各種ユーティリティサービス (Editor, IF, AND, OR, LOOP)を公開する．
- サイト管理者: Globusの提供するWS-GRAM (アプリケーション起動用)サービスとRFT (ファイル転送用)サービスを用意する．また我々が提供するパッケージを用いて，AI Factoryサービスと AIサービスを公開する．
- アプリケーション所有者: 各所有者が所属するサイトのAI Factoryサービスにアプリケーション情報を登録する．登録したアプリケーション情報は，Indexサービスに集められる．
- エンドユーザ: Indexサービスに収集されたアプリケーション情報とユーティリティサービスを用いて，ワークフローおよびデータフローを設計する．設計されたアプリケーション連携は，AIサービスおよびユーティリティサービス間のメッセージ通知により実現される．

2.3 アプリケーションの登録

アプリケーション所有者は，WS-GRAMサービスにジョブをリクエストするためのXML言語であるRSL (Resource Specification Language)を用いてアプリケーションの起動情報を記述する．記述したRSLファイルをAI Factoryサービスに登録することで，所有するアプリケーションを容易にWebサービス化することができる．図3に，アプリケーション所有者が記述するRSLファイルの例を示す．図3において，「AIS_」で始まる文字列はシステムが提供する変数であり，入出力ファイルのパス情報などを表現している．AI Factoryサー

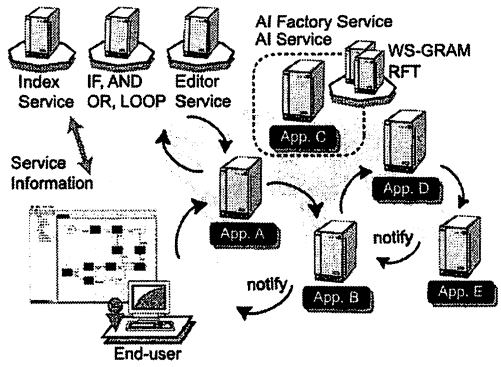

図 2: Application Igniting System の概要

```
<?xml version="1.0" encoding="UTF-8"?>
<job>
  <executable>/opt/blast/bin/blastall</executable>
  <directory>${AIS_JOB_HOME}</directory>
  <argument>-i</argument>
  <argument>${AIS_INPUT_FILE_PATH#0}</argument>
  <argument>-o</argument>
  <argument>${AIS_OUTPUT_FILE_PATH#0}</argument>
  <stdout>stdout</stdout>
</job>
```

図 3: RSL ファイルの例

ビスに登録されたアプリケーション情報は，すべて Index サービスに集められる．

2.4 リソースの生成とアプリケーションの実行

エンドユーザは Index サービスに収集されたアプリケーション情報を参照し，利用したいアプリケーションを複数選択する．アプリケーションの選択は，利用したいアプリケーション情報を保持する AI Factory サービスに，リソース生成を要求することで行なう．各 AI Factory サービスは，エンドユーザの要求に従い，登録されたアプリケーション情報を含むリソースを生成する．生成されたリソースには，AI サービスのポートタイプのみを通じてアクセスできる．AI サービスは生成された各リソースに対するメッセージ True を Producer から受け取ることで，アプリケーションの起動を開始する．アプリケーションの起動は以下の手順で構成される．

1. RFT サービスを通じ，あらかじめエンドユーザにより指定されたファイルをアプリケーションの入力ファイルとして取得する．

2. 全てのファイル転送終了後，リソースに含まれる RSL ファイルの情報を用いて，WS-GRAM サービスにアプリケーション実行をリクエストする．

3. アプリケーション実行後，新たなメッセージ Finished(正常終了) もしくは Failed(異常終了) を自身の Consumer に通知する．

2.5 アプリケーション連携の設計

エンドユーザは利用したいアプリケーションを選択し，複数のリソースを生成した後に，AI サービスのポートタイプを用いてアプリケーション連携を設計する．

2.6 ワークフローの設計

ワークフローの設計は，生成した各リソース間のメッセージ通知を構成し，アプリケーション実行の連鎖を決定することで行われる．Application Igniting System はアプリケーションの逐次実行や並行実行，同期実行や繰り返し実行を可能にするために，以下示すユーティリティサービスを用意している．

- IF サービス: エンドユーザにより定められた条件値に一致するメッセージを受け取った際に，新たなメッセージ True を通知する．
- AND サービス: 2 つの Producer に通知予約が行なえ，両者からメッセージ True を受けとった際に，新たなメッセージ True を通知する．
- OR サービス: 2 つの Producer に通知予約が行なえ，いずれかの Producer からメッセージ True を受けとった際に，新たなメッセージ True を通知する．
- LOOP サービス: メッセージ True を受け取った回数をカウントし，エンドユーザにより定められた回数に満たない場合にはメッセージ False を，同数の場合にメッセージ True を通知する．

IF サービスを用いたアプリケーション A，B の逐次実行の例を図 4 に示す．図 4 において，エンドユーザはアプリケーション A の情報を含むリソース A から IF サービスのリソースに，IF サービスのリソースからアプリケーション B の情報を含むリソース B にメッセージ通知を伝搬させ，また IF サービスにおいて条件値 Finished を指定してメッセージを True に変換するよう指示する．WSN において，Producer から Consumer にメッセージを通知するには，あらかじめ Consumer から Producer

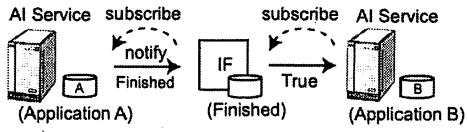

図 4: アプリケーションの逐次実行

へ，メッセージとは逆方向の通知予約を行なう必要がある．つまり，IF サービスのリソースはリソース A の Consumer となるため，エンドユーザは IF サービスのポートタイプを通じ，IF サービスからリソース A を保持する AI サービスに通知予約を指示する．またリソース B は IF サービスの Consumer となるため，リソース B を保持する AI サービスから IF サービスに，AI サービスのポートタイプを通じて通知予約を指示する．

2.7 データフローの設計

データフローの設計は，エンドユーザとアプリケーション間の入出力ファイル転送の他に，アプリケーション間の入出力ファイル交換を，AI サービスのポートタイプを通じて指示することで行なう．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場合，後者に該当するリソースを保持するサービスに対して入出力ファイルの組み合わせを指示し，アプリケーション起動の最初に入力ファイルとして取得させる．交換する入出力ファイルの書式が異なる場合には，ユーティリティサービスの 1 つである Editor サービスを経由してファイルの変換を行なう．Editor サービスは，エンドユーザの用意する変換用のアプリケーションを起動する機能を提供しており，エンドユーザはあらかじめ変換用アプリケーションを Editor サービスに提供しておくことで，書式の変換を実現できる．

2.8 クライアントツール

エンドユーザによるアプリケーション連携の設計は，我々が独自に定義したアプリケーション連携記述言語 (Application Integration Description Language: AIDL)，もしくは図 5 に示すクライアントツールを用いて行なう．特に図 5 のクライアントツールでは，リソース間のメッセージ通知やアプリケーション間の入出力ファイル交換を視覚的に表現でき，AIDL ファイルの自動生成を可能にする．さらにこのクライアントツールは，全てのリソース生成時に通知予約可能なリソースプロパティ全てに通知予約を行うことで，リソース間のメッセージ通知を自身にも派生させる．これにより，アプリケーション連携実行時の全てのメッセージ通知とその内容をモニタリングする．

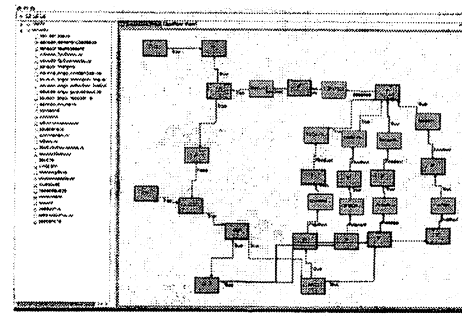

図 5: クライアントツール

3 性能評価

本章では，15 種類のアプリケーションを用いた構造解析へ Application Igniting System を適用し，システムの性能評価を行う．

3.1 アプリケーションの配置と実験環境

本性能評価では，ADVENTURE プロジェクトにより開発されたモジュール構造を有する 15 種類のアプリケーションを用いてシステムの性能評価を行なう．ADVENTURE プロジェクトは，設計用大規模計算力学システムの開発を目的としたプロジェクトであり，プロジェクトで開発された多数のアプリケーションをうまく組み合わせて利用することにより，構造や流体の解析，最適設計などが可能となる．本性能評価では図 6 に示すグリッド環境を用い，アプリケーション所有者を想定して「advauto_」で始まる 10 個のアプリケーションを Xenia クラスタに，その他の 5 つのアプリケーションを Galley クラスタに配置し，それぞれの AI Factory サービスに登録した．各アプリケーションは，WS-GRAM サービスにより起動され，ジョブスケジューラを経由してクラスタ内部の計算機で実行される．さらに，グローバル IP アドレスを割り当てた 3 つの計算機に，エンドユーザ，Editor サービス，ワークフロー設計を支援する 4 つのサービスをそれぞれ割り当てた．同一ネットワーク上に 3 つの計算機は接続されている．

3.2 ワークフローとデータフローの設計

エンドユーザの設計したワークフローを図 7 に示す．紙面の都合上，各アプリケーションおよび Editor サービスの後に付随する IF サービスの記述を省略した．全ての IF サービスに指定する条件値は Finished である．エンドユーザは各リソース生

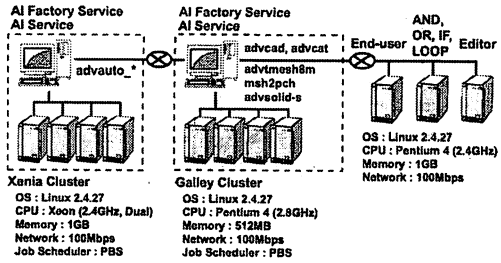

図 6: 性能評価に用いたグリッド環境

成後に，図 7 の矢印の逆向きに通知予約を指示する．また性能評価という観点から，LOOP サービスを用い，図 7 のワークフロー全体を 5 試行繰り返して平均の実行時間を測定する．アプリケーション連携を実行する際には，IF* に該当するリソースを保持する IF サービスからメッセージ True を通知するよう指示する．また AND* に該当するリソースを保持する AND サービスからメッセージ True を受け取ることで，アプリケーション連携の実行終了を判断する．

次に，エンドユーザの設計したデータフローについて図 8 に示す．図 8 において，黒く塗りつぶされた交点の左方向に位置するアプリケーションから，交点の下方向に位置するアプリケーションへの入出力ファイル交換を指示する．各入出力ファイル交換では，下方向に位置するアプリケーションの起動の際に，左方向に位置するアプリケーションの出力ファイルが，それぞれ 1 つずつ転送される．また，エンドユーザはアプリケーション連携実行前に，解析対象の構造物の形状を記したファイルを Editor(1) サービスに，材料特性を記述したファイルを advauto_single_constant2adv に送信する．さらに 3 つの Editor サービスに，それぞれ書式変換用のアプリケーションを配布する．アプリケーション連携実行後は，advauto_volume から構造物の体積が記述された出力ファイルを，advauto_equivstress から最大応力が記述された出力ファイルをそれぞれ受信する．

3.3 性能評価結果

図 7 のワークフローで用いられた 15 種類のアプリケーションの，WS-GRAM サービスを通じた総実行時間は 5 試行の平均で 146.3 秒であった．また図 8 のデータフローにおける，アプリケーション間のファイル交換の総実行時間は平均 146.8 秒であった．アプリケーション間のファイル交換は 1 試行につき 32 回行われ，RFT サービスを通じた

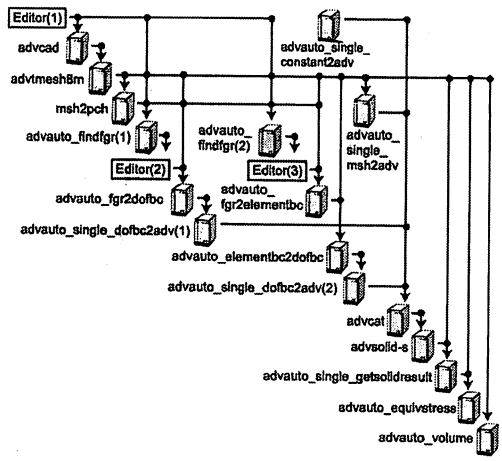

図 8: 構造解析のためのデータフロー

1 回のファイル交換に平均 4.6 秒かかっていることから，オーバーヘッド時間は小さくないことがわかる．また 3 回の Editor サービスによる書式変換の実行時間は 13.7 秒であり，平均 4.6 秒であった．一方で，これらの実行時間の和は 306.8 秒であるが，アプリケーション連携全体の平均実行時間は 229.3 秒であった．これは図 7 のワークフローにおいて，リソース間のメッセージ通知が複数のパスに分岐する部分があり，アプリケーション実行が並行して行なえたために，全体の実行時間が短縮できたものと考えられる．

また図 7 のワークフローにおいて，太い矢印で示されたメッセージパスは，並行して実行されるアプリケーション連携の最も実行時間のかかるパスを示している．このパスでは，WS-GRAM サービスを通じたアプリケーションの総実行時間は 5 試行の平均で 100.5 秒，アプリケーション間のファイル交換の総実行時間は平均 114.8 秒，2 回の Editor サービスによる書式変換の実行時間は 8.6 秒であった．これらの実行時間の和は 223.9 秒となり，メッセージ通知に関連するその他のオーバーヘッド時間は平均 5.4 秒と非常に短いことがわかる．

さらに本性能評価において，Application Igniting System を比較的大規模なアプリケーション連携に適用したことにより，次のような知見が得られた．まず，Globus(バージョン 4.0) における WSN の実装はメッセージの配送を保証していないが，本性能評価においては一部のメッセージ通知が失われたり，メッセージ内容に誤りがあるという現象は発生しなかった．そのため，WSN を基盤としたアプリケーション連携システムは，実用的な利用に

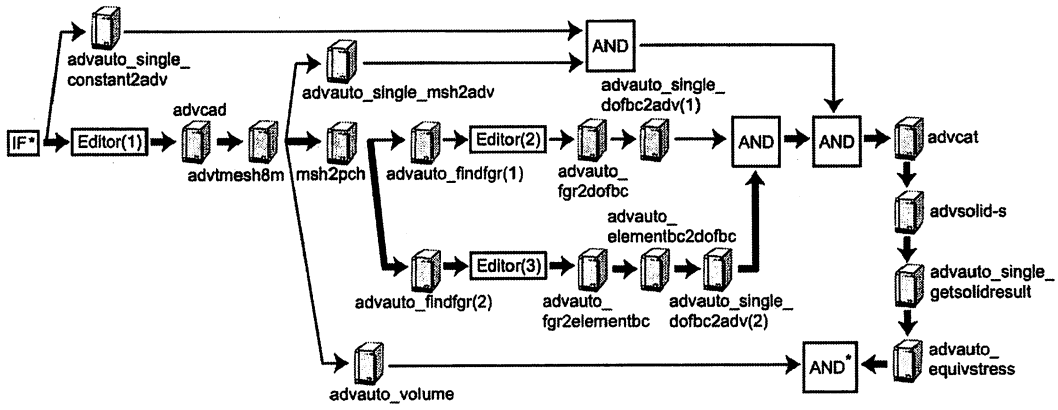

図 7: 構造解析のためのワークフロー

おいても問題はないと考えられる．一方で，本性能評価とは別に，1つのPCクラスタ上にアプリケーションを偏らせて配置した環境で実験したところ，偏らせて配置したPCクラスタのマスターノードの負荷が高まり，RFTサービスにおけるGridFTPの実行にエラーが発生した．これは，並行して実行されるアプリケーションの，アプリケーション実行前に行なわれる入出力ファイル交換が，同一ノード上で同時期に多数発生したことに起因すると考えられる．本実験では，GridFTPによるファイル転送をxinetd経由で行なっていたために，今後はデーモンとしてGridFTPサーバを起動しておくといったことを検討する必要がある．また，同一のAI Factoryサービスに登録されたアプリケーション間における入出力ファイル交換は，RFTサービスを経由しない，もしくはファイル転送自体を行なわないで，入出力ファイルのパス情報のみを交換するなどといったことについても検討する必要がある．

4 まとめと今後の課題

Application Igniting Systemは，WSNを基盤としたグリッド上のアプリケーション連携システムである．Application Igniting Systemでは，Webサービスとして表現されたアプリケーション間でメッセージ通知を伝搬させることにより，アプリケーション実行の連鎖を実現する．本研究では，ADVENTUREプロジェクトにより開発された15種類のアプリケーションを用いて，Application Igniting Systemを構造解析事例に適用し，その性能を評価した．その結果，比較的大規模なアプリケーション連携においても，Application Igniting Systemは実用的に利用可能であること，メッセージ通知に関連するオーバーヘッド時間は非常に短いことを明らかにした．一方で，アプリケーション間のファイル交換に要する実行時間が長く，今後の改善が必要であることがわかった．今後の課題として，RFTサービスを経由したアプリケーション間の入出力ファイル交換の回数削減や，グリッド用の分散ファイルシステムの導入などによる，RFTサービスに代わるアプリケーション間の入出力ファイルの交換方法について検討する必要がある．

参考文献

1) 下坂久司，廣安知之，三木光範: グリッド上のアプリケーション連携システムの提案と実装，先進的計算基盤システムシンポジウム SACSIS2006，pp.343-350 (2006).

2) 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`

3) ADVENTURE Project: `http://adventure.q.t.u-tokyo.ac.jp/`

4) Foster,I. and et.al.: The Open Grid Services Architecture, Version 1.0, Global Grid Forum OGSA-WG, GFD-I.030 (2005).

5) Czajkowski,K. and et.al.: The WS-Resource Framework, Version 1.0, (2004). `http://www.oasis-open.org/committees/download.php/6796/ws-wsrf.pdf`

6) Foster,I. and Kesselman,C.: Globus: A Metacomputing Infrastructure Toolkit, *International Journal of Supercomputer Applications*, Vol.11, No.2, pp.115-128 (1997).