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

2005－ARC－162 （4）
2005－HPC－101 （4）
2005／3／7

携帯端末から利用可能なグリッドのためのウェブフレームワーク

越本浩央，金澤正憲，岩下武史
京都大学大学院情報学研究科

グリッドコンピューティングの基盤を支えるミドルフレームワークが充実し，ウェブサービスとの統合による応用面が注目されている．しかし仕様の複雑さと規模の大きさが開発と利用を困難にしている．本研究ではこの二つの問題を解決するために，REST アーキテクチャに基づいたグリッドサービスの設計を提案し，モナドベースの RESTful ウェブサービスを構築する．REST アーキテクチャはネットワーク上のエンドポイントのステートレス化を推し進める．またモナドはデータと計算と計算戦略を切り離すことで計算結果への参照透明性を提供する．ここではモナドを利用することでの開発の効率化と RESTful なサービスの運用の利便性を示す．

A web framework for a grid computing on multi-platforms

Hiroo Koshimoto, Masanori Kanazawa, Takeshi Iwashita
Graduate School of Informatics, Kyoto University

With preparedness for the Grid computing by the Grid middle frameworks, a research and development of the Grid integration as the web-service are remarkable. However it seems too complicate on developing and operating, dues to a complexity and a fleshiness of the specifications. In order to solve these two problems, we proposed a efficiency of designing the Grid-service in line with the REST architecture, and composed a web-service based on Monad. The REST architecture makes endpoints on the Net stateless. Monad makes programs be decomposed into data, functions and strategies, and provides transparency of calculus. We show an efficiency of Monad for developing and a convenience of a RESTful web-service for operating.

1．　背景

Globus Toolkit[1], UNICORE[2][3]を始めとするグリッドコンピューティングのためのミドルフレームワークが充実してきた昨今，それらをベースとしたアプリケーション面に注目が集まっている. Globus はウェブサービスとしてグリッド技術を利用するための標準フレームワークとして WSRF[4]を公開している.

しかしこのようなフレームワークを利用する場合，サービスのインターフェースのみ WSDL として URL アドレスと共に提示されるだけで，サービスの利用や計算結果の再利用という段階ではポータビリティのある簡単で一意な参照が提供されていない．これは計算過程から結果参照までをステートフルなトランザクトとして実現しているためであり，携帯端末のように貧弱なブラウザ環境ではサービスの利用が不可能である．

また WSRF をはじめとして多くのグリッドサービスフレームワークは Java をベースとして構築されているが，実際に運用されている多くのウェブサービスが Perl/Ruby/Python などの Lightweight Language で実装されている．これはサービス開発を簡易化しターンアラウンドを高めるためであるが，グリッドサービスのためのフレームワークとしてこれを支援するものが欠けており，グリッドを専門としない一般的な開発者には手を出せない代物となっている．

以上二つの問題点を考慮して，本研究では携帯端末を含めた多様なプラットフォーム上から利用可能なグリッドサービス，及びその構築に求められるウェブフレームワークを説明し，一実装例を提案する．

2． RESTful グリッドサービス

グリッド技術の利用形態を広げる場合の問題としてステートフルなコンピューティングがある．適切に設計されたサービスにおいては特に問題とならないが，利用工程の柔軟性や利用端末の多様化を考えた場合には問題となる．これはグリッドサービスにアクセスした後の過程が端末環境に依存する点，ブラウザ側に多くの機能を要求する点などである．

Roy Fielding はウェブサービスに関してこのような問題を解決する REST アーキテクチャ[5]を提案している．REST アーキテクチャが明示しているのは簡潔に言えば次のようなものである．

- ウェブ上の公開されるべきリソース（ステート）全てに URI が割り振られる
- リソースは HTML で閲覧される
- HTTP を使う
- クライアントはブラウザでリソースにアクセスする（実体は HTML）
- サーバはアクセスされるリソースを用意する（方法は任意）
- ステートとしてサービスを実現
- ページの移動をステートの遷移と考えてサービスを構築する

つまりこれまで成長してきた WWW そのものであり，ベースの部分にはこれ以上の標準や拡張を施さない．このガイドラインにのっとってリソースの一意性を保持することにより，汎用性のあるサービスが構築可能である．

既に Globus はリソースアクセスのポータビリティのために RESTful SOAP の標準化を試みている．SOAP では任意のプロトコルを使用して全ての情報を包み込んだやり取りが行われ，エンドポイントに特定のステートを要求する．RESTful SOAP では HTTP の GET/POST/PUT/DELETE メソッドを利用してステートレスなウェブサービスを実現する．

本研究で提案するアプローチはよりラジカルなものである．HTTP のみで完全にステートレスな（RESTful な）グリッドサービスインターフェースを用意する．方針は以下の通りである．図.1 を併せて参照.

- 計算要求をステートとする
- 計算要求は URL として明示する
- ステートの状況は HTML で確認
- 計算結果は HTML として保持
- 計算の実行管理はサーバが行う

⇒再利用・汎アクセスを実現するグリッドサービス及び計算出力

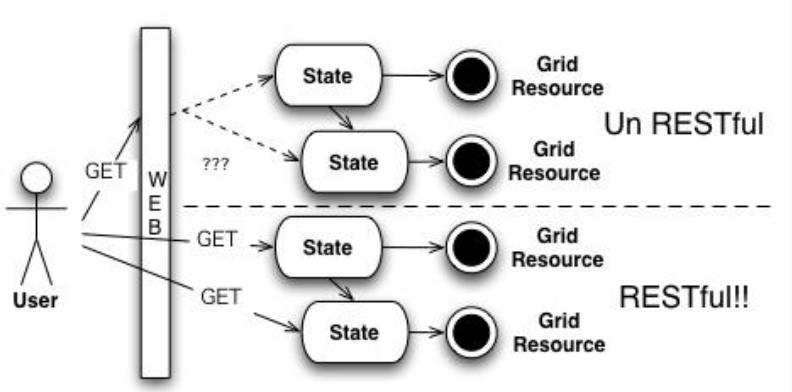

図.1 RESTful グリッドサービス

3. モナドベースアーキテクチャ

RESTful グリッドサービスの具体的な実装を考える場合，一意性を保って計算要求（要求後はサイト上のリソースと結びつく）と URL を一致させることが課題となる．計算に対して値の一致を保障して意味を与えることは参照透明性の問題であるが，解決策として Hakell でのモナド[6]が挙げられる．これは本来，破壊的代入を伴う命令型の処理に対して参照透明性を維持するためのもので，数学的には単位半群で右オペランドに関数をとる代数的構造である（だから正確には単位半群ではない）．簡単に言えば，モナドとは処理に相当する手続きとデータの間を補う糊であり，これを使うことで特定の処理とそれを要求する手順を分離することが出来る．

本研究ではモナドベースでリソース管理を行うフレームワークを実装した．URL とグリッドへの計算要求を対応付けるために，計算要求の記述をリスト.1 の形式に従って行う．これにより HTTP を利用した GET メソッドのみで計算プロセスの起動と参照が可能である．これは TCP/IP を話せる端末であれば最低限利用出来ることである．実行された計算プロセスは同じ URL でアクセスし，計算状況や結果を見ることが出来る．これらのデータはネットワーク上で保持出来るし，第三者への公開も可能である．図.2 参照.

リスト.1
計算要求の表現
http://FQDN/user/data?>>func(val)
FQDN: サービス提供サイト
user: サービス利用者
data: 計算対象となるデータ
func: 実行する計算（関数）
val: 呼び出す計算への引数
※>>func(val)は連続して記述して良い

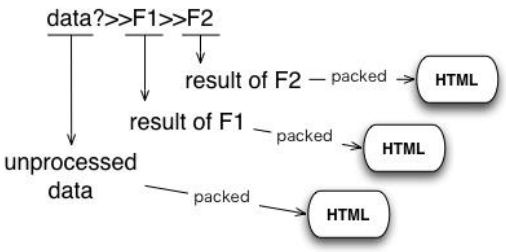

図.2 モナドベースアーキテクチャ

リスト.1 に示す通り，全ての計算要求はデータセントリックで定義される．細かな計算内容はサービス提供サイトが用意するものを利用し，>>演算子で接続する．計算工程のいずれの段階にも参

照透明性が保たれており，任意の工程を参照することも可能である．具体的な例は後述する．

4． 実装

Python を利用し本サービスを備えた簡易な HTTP サーバを実装した．サーバは GET される URL をパースし，内部的なデータベースに計算要求を記録した後, UNICORE サービスを起動する。次に計算要求に対応する HTML ファイルを作成する．データベースに登録された要求は定期的に状況の更新が行われ，対応した HTML ファイルが動的に書き換えられる．利用者は最初に要求した URL アドレスにアクセスすることで，自由にどこからでも状態や結果を参照することが可能である．

サービス提供者は，具体的なグリッドプロセスと対応したサービスの公開のために，参照要素（URL に記述できるもの）として使える識別子（関数）を用意しなければならない．この関数をアクセスする URL に記述すると，サーバは処理対象となるデータに関連付けられたスクリプトを実行し，具体的なデータを参照して実体のあるグリッドプロセスを起動し対応するページを提供する．一方参照されるサーバ上のデータには，任意の処理を受け付けて自らの表現を生成する雛形テンプレートが必要になる．これが（正確にはスクリプトの処理も含めて）モナドに相当する．データフォーマットごとにテンプレートの用意が必要となる．テンプレートは処理されたデータをグラフや表として定型的に（端的には HTML で）表現するための雛形である．処理されたデータは，この雛形テンプレートに埋め込まれて HTML ファイルとしてサーバ上に保持される．計算途中である場合なら経過などが記述され，定期的に更新される．概略は図.3 を参照.

雛形テンプレートは XML Scheme のように具体的なデータの表現形式を定めている．一方，これを処理するスクリプトは任意のテンプレートから必要となるデータの切り出しを行う。そのためグリッドプロセスの側ではデータの形式を意識せずに任せられた処理だけを達成すればよい．またスクリプトは計算結果を HTML ファイルとしてテンプレートに従いパッケージ化も行う．常にパッケージ化された HTML を介してサーバ上のやり取りが実行される. そして全ての段階の HTML がリソースとして（計算要求された）URL と結びつき公開される.

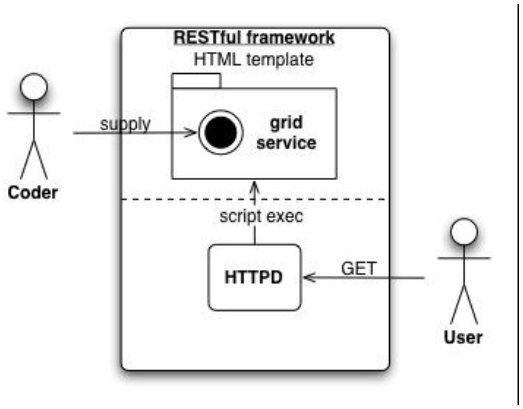

図.3 RESTful グリッドの実装

なおここで扱っているデータは事前にサーバが参照可能な領域に存在しなければならない．本実装ではユーザ単位で個別の 1 レベルな名前空間を用意しており，リスト.1 の場合であれば.data がユーザ user のディレクトリにファイルとして事前に存在している．データのアップロードは POST メソッドを用いて実装しているが，探して取ってくるというグリッドらしい機能はまだ備えていない.

5． 評価

サービスの利用例を示し，ユーザサイド・サーバサイドでどのような流れになっているかをリスト.2 に示す.

リスト.2

利用目的「応力解析がしたい」

Step1

User: 構造と応力のデータをアップロード

Server: POST されたファイルに対応したモナドを作成し，ファイルとして保存

model.xml → model.html

stress.xml → stress.html

Step2

User: メッシュの自動作成のため次を参照

model.html?>>auto_mesh('high')

Server: auto_mesh として登録されているサービスを model.xml をパラメータに与え'high'の設定で実行．稼動プロセスとしてウェブサーバのデータベースに登録し，model.html?>>auto_mesh で参照可能な HTML を作成

Step3

User: 三次元の有限要素法を適用

Model.html?>>auto_mesh('high')>>fem3d(stress.html)

Server: auto_mesh の出力を受けて fem3d を stress.xml で実行するプロセスを未処理で登録．次の更新で auto_mesh の終了が確認されると，未処理プロセスを投入．未処理時も HTML ファイルを用意し，状態を提示する．

Step4

User: 好きなときに好きなところからアドレスを参照出来る

元のデータが見たければ,

http://foo.ac.jp/User/model.html

メッシュは,

http://foo.ac.jp/User/model.html?>>auto_mesh('high')

解析結果は,

http://foo.ac.jp/User/model.html?>>auto_mesh('high')>>fem3d(stress.html)

以上をそれぞれ参照することで結果・状況が確認出来る．参照先には実体として HTML が存在するので, PC 端末だけでなく PDA や携帯電話でも確認が可能である．

Server: ゲットされる URL に対応した HTML をレスポンスする．端末によって簡易表示版などにリダイレクトしても良い．

データアップロードの段階を除けば，以上の作業は全て GET メソッドのみを備えたブラウザで可能である．これは非力な携帯電話でもグリッド技術が利用可能であることを示している．

一度サーバにアップロードされたデータはすべて HTML ファイルとしてパッケージ化されるので，計算要求する場合のデータ参照もそれに合わせる．こうすることで，サービス提供側が用意した枠組みで表示方法を決定したり，グリッドプロセスへ引き渡したりすることが出来る．内部実装についてユーザが気にかけることは無い．またサービス提供者は，提供するプロセスや扱うデータを限定して HTML へのパッケージ処理を実装するだけで良い．両者の依存性を絶ちつつ，相互運用性を実現するモナドの効用が確認出来る．

6. むすび

簡易で柔軟性のあるグリッドサービスフレームワークとして, RESTful なモナドアーキテクチャを示した．端末側への要求を減らすことで汎用性と計算の再利用が達成出来る．同時にデータセントリックな手法により両者の依存関係を解き，グリッドサービス拡張の手順も簡単化出来る．

本フレームワークの課題として, Globus toolkit など他のフレームワークへの対応，多様なデータパッケージのサポートと処理プロセスの提供，資源要求戦略の実装などが挙げられる．また計算工程・結果へのリンクを集約して，それを検索が可能なインターフェースを用意することで，より利

便性に優れたオープンなグリッドサービスを構築することが可能であろうと思われる．

一方で，今回はサービスサイトにデータを実体化する処理に古典的なものを用いた．連携するサイト間でデータを共有・検索し，よりシームレスで特定のアプリケーションやプロトコルを意識させることなくデータの参照を実現する仕組みが必要である．このためには，計算要求に利用するデータに対してよりグローバルに一意性が確保出来る識別子を与えることが重要であると考える．

参考文献

[1]Ian Foster, Carl Kesselman, Jeffrey M.Nick, Steven Tuecke “The Physiology of the Grid” DRAFT
http://www.globus.org/research/papers/ogsa.pdf

[2]Dietmar W.Erwin “UNICORE – A Grid Computing Environment”

[3]Michael Rambadt, Philipp Wieder “UNICORE – Globus: Interoperability of Grid Infrastructures”

[4]”Web Services Resource Framework” IBM http://www-106.ibm.com/developerworks/library/ws-resource/

[5]Roy Thomas Fielding “Architectural Styles and the Design of Network-based Software Architectures”

[6]Philip Wadler. “How to declare an imperative” *ACM Computing Surveys*, 29(3):240--263, September 1997.