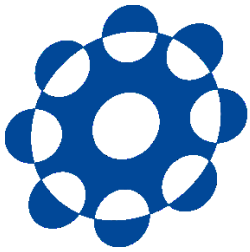
異種グリッド間インターオペレーション技術に関する研究

佐賀　一繁

博士（情報学）



総合研究大学院大学
複合科学研究科
情報学専攻

平成 24 年度
（2012 年）

2012 年 9 月

本論文は総合研究大学院大学複合科学研究科情報学専攻に
博士（情報学）授与の要件として提出した博士論文である。

審査委員：

合田 憲人（主査）

鯉渕 道紘

松岡 聡　　東京工業大学

三浦 謙一

漆谷 重雄

（主査以外はアルファベット順）

Interoperation among Different Type Grid Computing Middleware

Kazushige Saga

DOCTOR OF
PHILOSOPHY

Department of Informatics,
School of Multidisciplinary Sciences,
The Graduate University for Advanced Studies (SOKENDAI)

September, 2012

A dissertation submitted to
the Department of Informatics,
School of Multidisciplinary Sciences,
The Graduate University for Advanced Studies (SOKENDAI)
in partial fulfillment of the requirements for
the degree of Doctor of Philosophy

Advisory Committee :
Kent Aida（Chair）
Michihiro Koibuchi
Satoshi Matsuoka　　Tokyo Institute of Technology
Kenichi Miura
Shigeo Urushidani
（Alphabetical order of last name except chair）

概要

現代の科学研究は大規模化、高精度化、複雑化、多角化する一方であり、その円滑な推進にはコンピュータは欠かせないものとなっている。このため、ユーザからのコンピュータの大規模化、高速化への要望は限りがなく、個々の組織で対応するのが難しくなってきた。そこで、ユーザが必要とするコンピュータ資源を必要なときに必要なだけ得られる環境を目指し、組織の垣根を越えてコンピュータ資源を連携させるグリッドコンピューティング（以下グリッドと略）が考案された。この仕組みを利用すれば、ユーザは広域なネットワーク上にある複数の実組織や個人が所有するコンピュータ資源群やストレージ資源群を、あたかもひとつの仮想的な組織（Virtual Organization: VO）が所有する資源群であるかのように横断的に使用することが可能となる。また、グリッド環境はコンピュータ資源を補完するための連携に使われるだけではなく、研究コミュニティでデータやアプリケーションを共有し研究を円滑に推進するためにも使われている。現在、欧州、米国、中国などの諸外国では、グリッドコンピューティング技術による資源の組織間連携の有効性が認められ、国策としての大規模グリッド環境や様々な研究コミュニティによるグリッド環境が構築され広く利用されている。

このように、その有効性が認められ諸外国では盛んに利用されるようになってきたグリッド環境であるが、この技術が提案された当初にはグリッドとしてのインターフェースや情報表現方法などに標準仕様がなく、多くのグリッドミドルウェアが独自仕様で開発された。このため、異なるグリッド間では資源を連携させることができず、国際的な共同研究や異分野間の共同研究の円滑な推進に問題が出てきた。そこでグリッドコンピューティングの標準仕様を策定する団体 Open Grid Forum（OGF）では、異なるグリッド間でジョブを相互投入するための仕様 High Performance Computing Basic Profile（HPCBP）を策定した。多くのグリッドプロジェクト、大学、企業が HPCBP 準拠のジョブ実行サービスやモジュールを開発し、実証実験などによりその有効性を確認している。しかし、HPCBP 仕様で実運用レベルのインターオペレーション環境を構築するには多くの問題が存在しており、これを解決していく必要がある。

本研究は大きくは二つの課題からなる。第一にインターオペレーションに必要な機能や情報を明確にし、インターオペレーションアーキテクチャを提案する。さらに、HPCBP 仕様に準拠したミドルウェアを実装し、これを用いたインターオペレーション研究環境を構築して他グリッドとの実証実験を行う。第二にインターオペレーションにおいて発生が予想されるコンピュータ資源間のロードバランスの乱れによる性能低下問題と、ユーザアカウント管理方法の違いによるデータステージングの回数増加問題を明らかにし、解決方法の提案と実験やシミュレーションによる有効性の確認を行う。

第一の課題であるインターオペレーションアーキテクチャの提案とは、単一グリッド環境と同じレベルの資源連携を、複数の異種グリッド間で実現するアーキテクチャを提案することである。この資源連携には、ジョブ投入・実行に関する仕様だけでなく、グリッド間の情報交換仕様や認証仕様など、さまざまな機能仕様や情報表現仕様が必要になる。これら仕様を明確にするには、まず、単一グリッドでのジョブ実行に必要な仕様を明確にし、単一グリッドでのジョブ実行の流れと、インターオペレーションにおけるジョブ実行の流れとを比較してインターオペレーションとして必要な仕様を明確にする。さらに、これを現状唯一適用可能である HPCBP 仕様に当てはめ、不足している機能仕様や情報表現仕様などを明確にする。実際には HPCBP はジョブ投入・実行に関する仕様でしかなく、多くの仕様が不足している。例えば、グリッド間の資源状況の交換仕様やジョブのデプロイメント仕様などである。これらの仕様が不足することにより発生する問題を明確にし、その解決策を提案する。また、HPCBP への準拠は一般に既存のミドルウェアに追加する形で実現されることが多く、このとき既存仕様との整合性の問題が発生する。本研究では、国立情報学研究所が開発した NAREGI ミドルウェアの HPCBP 準拠実装を通じ、これらの仕様上の問題に対する解決策を提案する。また、この実装を使用してインターオペレーション研究環境を構築し、諸外国のグリッドと実証実験を行ったので、その結果を報告する。

第二の課題の第一の問題は、クライアント、メタスケジューラ、コンピュータ資源などからなる一組の完結したグリッド環境（以下ローカルグリッドと略）に、他グリッドからジョブが投入されることにより、ローカルグリッドのコンピュータ資源間のロードバランスが乱れ、実行待ちジョブの増加、実行待ち時間の長期化などの問題が発生すると予想されることである。これは、他のグリッドが予告なしにローカルグリッドのコンピュータ資源に直接ジョブを投入することにより発生する。まず、この問題が実際に発生することを実験とシミュレーションによって示し、問題の原因を検討する。そして、この原因に対する解決策を提案し、シミュレーションによってその有効性を確認する。次に第二の課題の第二の問題は、コンピュータ資源のユーザアカウント管理方法の違いに起因するデータステージング回数の増加問題である。グリッド環境のコンピュータ資源のユーザアカウントとその作業領域の管理方法には、単体コンピュータシステムのような静的なユーザアカウント管理と、ジョブ毎に動的にユーザアカウントを割当てる一時ユーザアカウント管理の 2 種類の方法がある。2 つのコンピュータ資源で依存関係のあるジョブを実行した場合を考える。各資源上のジョブの間でファイルのデータステージング（データ転送）が必要となるが、一時アカウント管理のコンピュータ資源間の方が、アカウントや作業領域のライフタイムの都合で、静的アカウント管理のコンピュータ資源間よりデータステージングを 1 回多く必要とする。異なるアカウント管理のコンピュータ資源を混在して使用する場合、一時アカウント管理のデータステージング方法を採用せざるを得ず、静的アカウント管理のグリッドユーザから見ると、データステージングが 1 回多く必要となり、その時間だけ

ジョブの実行に時間を要することになる。近年はデータが巨大化する一方であり、この時間は無視できないほど大きくなってきている。また、同じデータを複数回転送することはネットワークトラフィックの増大をもたらし、他のジョブへの影響することも考えられ望ましくない。本研究ではこの問題を解決する方法を提案し、実際にプロトタイプ実装を行って有効性を確認したので報告する。

Abstract

Because the modern scientific researches tend to large scale, high resolution, complicated, and diversified, computer systems are indispensable for the smooth promotion of them. Scientific users always want to use more large and more powerful system to solve their issues. This demand became too big to accept by individual organizations. To solve this issue, the grid computing was proposed. The grid computing middleware can federate the resources owned by individual organizations, as the resources owned by a virtual organization (VO). The users can use necessary amount of these resources when they want, like they use the resources which owned by their real organizations. The grid computing is used not only for complementing insufficient resources but also for sharing data and applications in research communities. The US, Europe, China and many countries recognized the effectiveness of resource federation among individual organizations, and they built many kinds of national and community gird infrastructures.

When development of the early grid middleware began, there were no standard specifications. These middleware were developed using proprietary specifications. Therefore, it was impossible to federate resources among the different type grid environments. Many latest grid middleware are still based on many proprietary specifications. This makes problems for smooth promotion of collaborations of different research area and international collaborations. To solve the problem, the Open Grid Forum (OGF), an organization which defines grid standards, defined "High Performance Computing Basic Profile" (HPCBP) specification for mutual job submission among different type grid environments. Many academic grid projects and companies developed the HPCBP compliant service and client, and confirmed the effectiveness of the specification. However, the HPCBP specification has many issues to apply it on production level grid infrastructures.

This paper focuses two kinds of issues, architectural issues and performance issues. One of the goals of architectural issues is to propose an interoperation architecture which has the same functional level resource federation with single production level grid environment. At first, to make necessary specifications for the production level grid interoperation clear, this paper made the necessary functions and information in each step of job submission process on a single grid environment clear. And then, the necessary functional specifications and information specification for interoperation were discussed. Then the HPCBP specification was compared with them, and was made

insufficient specifications clear. Since the HPCBP is just only a specification for job submission management, there are many insufficient specifications. For instance, resource information exchange specification and application deployment specification are not defined. This paper showed the problems caused by those insufficient specifications, and proposed solutions. Usually, the HPCBP service and client are implemented as an extra function in the conventional grid middleware. Therefore, many consistency problems will occur between the conventional specifications and the HPCBP specifications. This paper proposes solutions of this issue, through implementation of the HPCBP compliant service and client to the NAREGI grid middleware which was developed by the National Institute of Informatics. Then a grid environment was built using this middleware and interoperation conformance tests with other type grid environments of foreign countries were performed.

Also this paper focused two performance issues, the workload balance disruption issue and the data staging issue. These problems are expected to occur in interoperation environments. A set of single grid environment, which is called “local grid” in this paper, consists of client, meta-scheduler, computing resources, and so on. The local grid meta-scheduler submits jobs, which come from the local grid client, into the local grid computing resources according to their workloads. Because the jobs from other grids are submitted to the local grid computing resource directly, not through the local grid meta-scheduler, the workload balance is expected to be disrupted. The workload disruption problem will be the cause of increases of waiting jobs and job running time. This paper showed that the problem will occur by an experiment and a simulation, and analyzed the cause of the problem. And then, this paper proposed the solution of the problem, and showed the effectiveness of it by a simulation.

Another performance issue is the data staging issue. In the grid environments, there are two types of user account and its working area management method. The static user account management method assigns user account and allocates its working area, such as home directory, statically using the same way with conventional computer systems. The temporal user account management method assigns a user account and allocates its working area dynamically when the computing resource receives a job. The lifetimes of the user account and its working area are equal with the job running time. In the case of jobs with dependency, the result file of the first job has to be transferred from the first job resource to the second job resource. If the both resources are managed by the static method, the file can be transferred from the first job resource to the second job resource directly. On the other hand, if the both resource are managed by the temporal method,

because the lifetime of the user account and the working area of the both jobs are not overlapped, the file transfer between the both resources is done with two data transfer steps, file transfer from the first job resource to the temporal storage and the file transfer from the temporal storage to the second job resource. When the account management type of the both resources are different, because of the lifetime of the user account and some restrictions, the same file transfer method with the temporal method should be used. Thus, if the temporal account resource is used as the source or the destination resource, an extra file transfer, compared with the file transfer between static account resources, is required. In recent days, because the computing data tend to become huge, the transfer time and the network traffic of this extra file transfer are not negligible. This paper proposed a solution for this problem, and showed the effectiveness of the proposal by two experiments on a prototype system.

表記について

本論文はグリッドコンピューティング技術に関する研究論文である。グリッドコンピューティングに関する論文や記述では、コンピュータ、ストレージ、ミドルウェア、アプリケーション、データ、データベースなど、ユーザが目的とする計算を実行するのに必要な様々な資源が議論される。これらの議論の中で、たびたび「計算資源」という用語が用いられるが、その意味するところが議論によって異なる。コンピュータのみを指す場合と上記目的の計算を実行するために必要な資源全体やその一部を表す場合など様々ある。本論文では混乱を避けるために、以下のように用語を定義して使用する。

- **計算資源**

コンピュータ、ストレージ、アプリケーション、データ、データベース、さらにはこれらを管理するグリッドミドルウェアからなる、ユーザが目的の計算を実行するための資源全体を指す。

- **コンピュータ資源**

上記計算資源のうち、コンピュータとこれをグリッドコンピューティング環境の資源としてサービスするためのミドルウェアを指す。

- **ストレージ資源、グリッドストレージ、ローカルストレージ資源、ローカルストレージ**

「ストレージ資源」と「グリッドストレージ」は同じものを意味し、グリッド上で共有されるストレージを指す。上記計算資源のストレージと、これをグリッドストレージ資源としてサービスするためのミドルウェアからなる。一方、各サイト内のみで共有され他のサイトからアクセスできないストレージやコンピュータ資源に直接接続されているストレージなど、グリッド上のストレージとしてサービスを行っていないストレージは、「ローカルストレージ資源」または「ローカルストレージ」と区別して表記する。

目次

第 1 章　　異種グリッド間インターオペレーション

本章では、本研究の課題である異種グリッド間インターオペレーション（以下、インターオペレーションと略）について、まずその背景にあるグリッドコンピューティング発展の概要とインターオペレーションが検討された経緯を説明する。次にインターオペレーションを実現するための技術として国際標準仕様の概略と抽象的インターオペレーションアーキテクチャを説明し適用例を示す。さらに、標準仕様である HPCBP の問題点とインターオペレーション環境で発生が予想される問題について説明し、本研究の目的と貢献について説明する。最後に本論文の構成について説明する。

1.1.　背景

● 組織を跨る資源連携

現代の科学研究は大規模化、高精度化、複雑化、多角化する一方であり、その円滑な推進には計算機は欠かせないものとなっている。このため、ユーザからの計算資源の大規模化、高速化への要望は限りがなく、個々の組織で対応するのが難しくなってきた。そこで、ユーザが必要とする計算資源を、必要なときに必要なだけ得られる計算環境を目指し、組織の垣根を越えた計算資源の連携や利用を可能とするグリッドコンピューティン技術が考案された [1] [2]。グリッドコンピューティング技術とは、広域なネットワーク上にある複数の実組織や個人が所有する計算機やストレージを、あたかもひとつの仮想的な組織

（Virtual Organization: VO）が所有する資源であるかのように、横断的に、安全に、使いやすく利用することを可能にする技術である。数多くのプロジェクトがグリッドコンピューティングを実現するためのミドルウェアを開発し、これを VO に参加する実組織の計算機やストレージにインストールすることにより、多くのグリッドコンピューティング環境（以下グリッド環境と略）が構築されてきた。グリッドコンピューティングには、スーパーコンピュータなど計算能力の高い計算サーバを連携させジョブ要件に適したコンピュータ資源を使用する方法から、非常に多数の個人所有の PC を連携させ特定目的のために必要な計算能力を得る方法まで [3] 様々な資源レベルと連携方法があるが、本論文では前者の計算サーバレベルのコンピュータ資源の連携を課題としている。このため本論文では、単にグリッドコンピューティングと記した場合は、この計算サーバレベルの連携によるグリッドコンピューティングを指す。

欧米では1990年代の後半からグリッドミドルウェアの開発とグリッド環境の構築が始まり、その有効性が認められて参加組織やコンピュータ資源量の拡大が続いている。そしてグリッドミドルウェアの拡張や改善、そのときどきのインフラ要件に合せた運用組織の更新などを行いながら現在に至っている。その形態は単一組織の複数部門の計算センターが所有するコンピュータ資源の連携といった私的なものから、大学や公的研究機関が持つ大規模計算環境を連携させたものまで、環境要件も規模も様々である。大規模なグリッド環境の代表としては、米国の TeraGrid [4]、Open Science Grid（OSG） [5]、FutureGrid [6]、欧州の Enabling Grids for E-sciencE（EGEE） [7]、Distributed European Infrastructure for Supercomputing Applications（DEISA） [8]などが有名であるが、TeraGrid は 2011 年にアーキテクチャと運用組織を更新して Extreme Science and Engineering Discovery Environment （XSEDE） [9] となり、EGEE と DEISA は 2010 年に欧州の他の主要グリッド環境である NorduGrid [10]、D-Grid [11]、欧州各国のナショナルグリッドとともに European Grid Initiative（EGI） [12]として統合され現在も更新や拡大を続けている。特に EGI は、今日では 300 以上ものコンピューティングセンターを結び、一万人以上の利用者が一カ月に 1,300 万ジョブを投入する非常に大規模な世界最大級のグリッド環境となっている [13]。

グリッド環境を構築する目的は、単に計算能力の不足を補うためだけではない、研究コミュニティにおいて、その円滑な研究促進のためにデータやアプリケーションを共有するためにも使用されている。例えば、EGI に統合された世界レベルの大規模グリッド環境 EGEE は、CERN を中心とした高エネルギー分野の共同研究において、コンピュータ資源やストレージ資源を連携するだけでなく、加速器から収集されたデータや研究アプリケーションを共有するために構築された。今日ではこの他に、天文研究関係、バイオ関係など、数多くの研究コミュニティがグリッド環境を構築している。

● **グリッド間資源連携への取り組み**

このように、欧米ではグリッド環境は共同研究環境として有効であることが認められ、広く利用されている。しかし近年、このような連携環境を利用する上でいくつかの障壁が問題視されており、これを解決するための研究が進められている。本研究課題であるグリッド間連携もそのひとつである。この他に、ユーザのローカルシステムとグリッド環境の連携の障壁、グリッド環境とクラウドコンピューティング環境（以下クラウド環境と略）の連携の障壁などもある。これらについては第 2 章の関連研究にて紹介する。

現在のグリッド環境を構成するグリッドミドルウェアは、研究コミュニティ、大学、国策プロジェクトなどにより 10 数年前から開発され、機能拡張や改善を繰り返ししながら使用されているものが多い。これらのグリッドミドルウェアの開発当初は、グリッドとしてのアーキテクチャ、インターフェース、情報表現などの標準仕様がなく、各プロジェクトとも独自のアーキテクチャ、独自の仕様で研究、開発を行っていた。このため、異なるミドルウェアで構築されたグリッド環境間では、インターフェース仕様の違いからコンピュータ資源やストレージ資源を連携させることができなかった。そこで、多くの他の従来技術と同様に仕様の標準化が叫ばれ、グリッドコンピューティング仕様の標準化団体 Grid Forum (GF、前 Global Grid Forum: GGF、現 Open Grid Forum: OGF [14]）が設立された。GF/GGF/OGF ではグリッドやクラウド関連の様々な機能のインターフェース仕様やその情報表現の標準仕様が議論、策定され、多くのグリッドプロジェクトがその仕様に準拠した。しかし、各グリッド固有のアーキテクチャ、特長とする機能、設計思想などの違いから独自仕様も多く残り、また同様な機能に複数の標準仕様がある場合の選択の違いなどから、異なるグリッド環境間ではインターオペレーションを実現できなかった。そこで OGF は、異なるグリッド環境間でジョブを投入する際に準拠すべき機能仕様を規定したプロファイル仕様 HPC Basic Profile (HPCBP) [15]を策定した。だが、HPCBP が準拠を規定する機能範囲（仕様）の小ささから、多くのプロジェクトや企業が HPCBP 仕様のサービスやモジュールを開発したものの、実運用レベルのインターオペレーションの実現には至っていない。各グリッド環境が実現している柔軟なジョブ投入やユーザ管理などは実現できず、相互ジョブ投入の単純なテストレベルにとどまっている。そこで現在、OGF では実運用に耐えうる次世代のインターオペレーション仕様 Production Grid Infrastructure（PGI） [16]の策定を行っている。なお、EGI や XSEDE では異種グリッド環境との資源連携を重視したアーキテクチャとなっており、独自の拡張をしながらも HPCBP 仕様でもジョブ投入できるサービスを実装している。このように、世界のグリッド環境は「グリッドのグリッド」に向っており、「必要なときに必要な計算パワーを使用できる」というグリッドの理想にさらに近づきつつある。

● 異種グリッド間インターオペレーション検討の経緯

異種グリッド間インターオペレーションは、2005 年に米国シアトルで開催された国際学会 SuperComputing05（米国、シアトル）の際に、TeraGrid（米国）の Charlie Catlett 氏、NAREGI（日本） [17]の松岡聡氏らが、世界中の多くのグリッドプロジェクトに呼びかけ会議を持ったのが公式な検討の始まりであると考えられる [18]。このときの会議には、世界各国から 11 の主要グリッド関連プロジェクトが参加した（APAC、BusinessGrid、DEISA、EGEE、GGF、K*Grid、NAREGI、OSG、Pragma、TeraGrid、UK NGS)。異種グリッド間の連携の重要性は誰でもが認めるところであったが、この時点では各グリッドプロジェクトとも独自仕様のインターフェースで開発、運用しており、対応は難しい状況であった。しかし、対応が遅れると各グリッドの複雑さが増加し、さらにインターオペレーションが難しくなる一方のため、直ちに行動を起こす必要があった。そこで、まず全てのグリッド間ではなく、インターオペレーションできそうなグリッド間でアイランドと呼ばれる少数グリッド間インターオペレーションを実現し、その次の段階としてアイランド間を結んで全グリッド間のインターオペレーションを実現する方針とした。この議論を受け、著者は 2006 年に国立情報学研究所を中心に開発がすすめられていた NAREGI ミドルウェアと欧州の CERN が開発･運用していた gLite [19]との間で独自仕様によるインターオペレーション機能開発し、その年に開催された SuperComputing06（米国、タンパ）にて、まず NAREGI から gLite にジョブを投入するインターオペレーションのデモを実施した。独自仕様による小規模インターオペレーションは、NAREGI-gLite 以外にも EU-China、EU-India などいくつかのプロジェクトで実施されたが、アイランド間を結ぶまでには広まらなかった。

インターオペレーションの議論は GGF（当時）でも検討が始まり Grid Interoperation Now Community Group（GIN-CG）が設立された。また、GGF ではアイランドだけではなく、その当時議論されていた仕様を組み合わせてインターオペレーションを実現するプロファイルファイル仕様 HPCBP が策定され、いくつかのグリッドプロジェクトや企業がプロトタイプを実装した。これも SuperComputing06 などでデモが実施された。その後もインターオペレーション活動は OGF の GIN-CG を中心によって継続され、現在に至っている。しかし HPCBP は上記のようにグリッド環境を運用する上での様々な機能が不足しているため、OGF では PGI 仕様の策定を推進している。なお、著者は 2010 年から 2012 年にかけて GIN-CG の共同議長を務め、インターオペレーションの実証実験などを推進した。

1.2.　異種グリッド間インターオペレーションの技術

1.2.1.　標準仕様

上記のように、グリッドコンピューティングのインターフェースや情報表現などの仕様の多くは OGF で策定されている。OGF は非営利の団体であり、多くの大学、研究所、企業の研究者や技術者が議論をして仕様を策定している。参加資格は特になく、誰でもが自由に意見を述べたり、新たな仕様を策定したりすることができる。ただし、新たな仕様策定にはチャーターと呼ばれるプロポーザルの作成とその承認プロセスが必要であるため、重複するような仕様や不要な仕様が策定されることはない。検討方法はワーキンググループにより異なるが、メーリングリスト、電話会議での議論、年に 3 回の会議（OGFnn と称される: nn は通算回数）によって策定されている。基本的に議論の資料や議事進行状況はインターネット上に公開しており、策定された仕様もインターネット上でコメントを募集し、コメントを反映したものが最終的な推奨仕様提案ドキュメントとなる。この段階ではあくまでも提案であり推奨仕様ではない。複数の実装で互換性が確認されたときに推奨仕様となることになっている。OGF のインターフェース仕様の多くは Web サービスレベルの仕様であり、その下位の通信や認証などのレイヤーの仕様は、W3C や OASIS など他の機関が策定した仕様に準拠している。2012 年 4 月 15 日現在、コメント募集段階の仕様を含め 48 の推奨ドキュメントが登録されており、その内訳はアーキテクチャ関連：8、ジョブ投入関連：10、データ関連：12、セキュリティ関連：3、API 関連：8、管理：2、その他：5 となっている。また、この他にも関連情報、サーベイ、実証実験結果なども公開している。

上記のように HPCBP 仕様は、特にジョブの相互投入を実現するために準拠すべき仕様を規定したプロファイル仕様である。具体的には、ジョブ記述仕様、ジョブスケジューリング・管理仕様、セキュリティ（認証クレデンシャル）仕様からなる。実運用レベルのグリッドでは、グリッド内の単純なジョブ投入でも更に多くの仕様を必要としており、HPCBP が準拠を規定した仕様だけで実運用レベルのミドルウェアを実装することはできない。

1.2.2.　HPCBP 仕様によるインターオペレーションアーキテクチャ

- **単一グリッドのアーキテクチャ**

インターオペレーションアーキテクチャを説明するために、まず単一グリッドのアーキテクチャを説明する。図 1-1 に単一グリッドの代表的なアーキテクチャを示す。一般的に 3 階層になっており、ユーザがジョブ投入や管理を行うクライアントやポータルなどの層、ユーザのジョブや操作をグリッド資源へ仲介するグリッド資源アクセスサービス群の層、計算処理を実行するコンピュータ資源とグリッドの共有ストレージであるグリッドストレージ資源の資源層からなる。ミドルウェアとしての実装方法は様々で、この層分けが絶

対的な訳ではない。ジョブ投入・実行の流れは次のようなものである。ユーザは認証クレデンシャル（信用情報）を使用してクライアントやポータルにログインする。ユーザはここでジョブを投入するためのドキュメントを作成し、投入をクライアントに指示する。図では要件の異なる 3 種類のジョブを投入している。投入されたジョブは、グリッド資源アクセスサービス層のメタスケジューラに転送され、メタスケジューラは資源ブローカを利用して各ジョブの要件に合致するコンピュータ資源を探し、これにジョブを投入する。ジョブの入力データや出力データはファイルとしてグリッドストレージとの間で転送される。

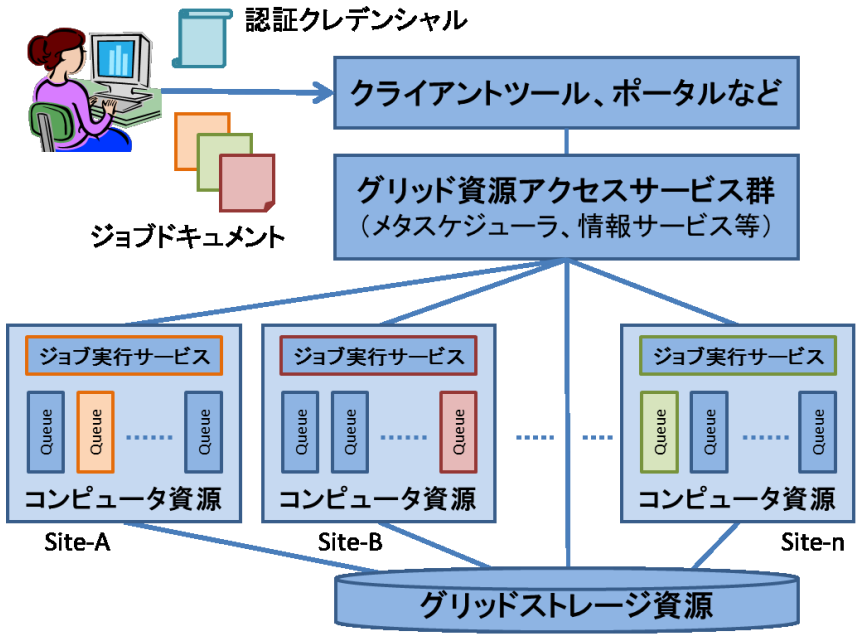


図 1-1　単一グリッドの代表的アーキテクチャ

● **インターオペレーションアーキテクチャ**

HPCBP は準拠すべきジョブ記述仕様、ジョブ実行サービス仕様、認証クレデンシャル仕様を規定しているが、これを何処にどのように実装するかは規定していない。しかし、ジョブ実行サービス仕様を何処に実装するかによって、インターオペレーション構造は大きく変わる。さらに、多くのグリッドミドルウェアでは従来互換性を保つために、独自仕様のジョブ実行サービスと HPCBP 準拠のジョブ実行サービスを併設する場合が多い。このため、アーキテクチャのバリエーションはさらに多くなる。HPCBP 仕様インターオペレーションの代表的な概念アーキテクチャを図 1-2 に示す。

図中の色の違いはグリッドミドルウェアの違いを示しており、この図は 3 種類のグリッ

ドミドルウェアで構築されたグリッド環境間のインターオペレーションを表している。Grid-A は HPCBP 準拠ジョブ実行サービスをコンピュータ資源に実装した例であり、HPCBP の機能不足や従来互換のために、コンピュータ資源に従来の独自仕様ジョブ実行サービスに HPCBP ジョブ実行サービスを追加している。この構造は比較的容易に実装可能であり、HPCBP プロトタイプ実装によく用いられている。Grid-B はメタスケジューラなどの上位層に HPCBP サービスを実装する例であり、後述の一部の本格的な運用グリッドでも採用されている。実装は比較的容易であるが、従来ミドルウェアからの移行では、従来環境で実現していた HPCBP にはない独自仕様による機能をどのように補うかが問題となる。Grid-C はグリッドアクセスサービス、コンピュータ資源の両方に HPCBP ジョブ実行サービスを実装する構造である。これは、クライアントからコンピュータ資源へ直接のジョブ投入とブローカなどのサービスを経由してジョブを投入する場合の両方のユースケースに対応できる。その他の組合せも含め、各構成の特徴を表 1-1 に示す。

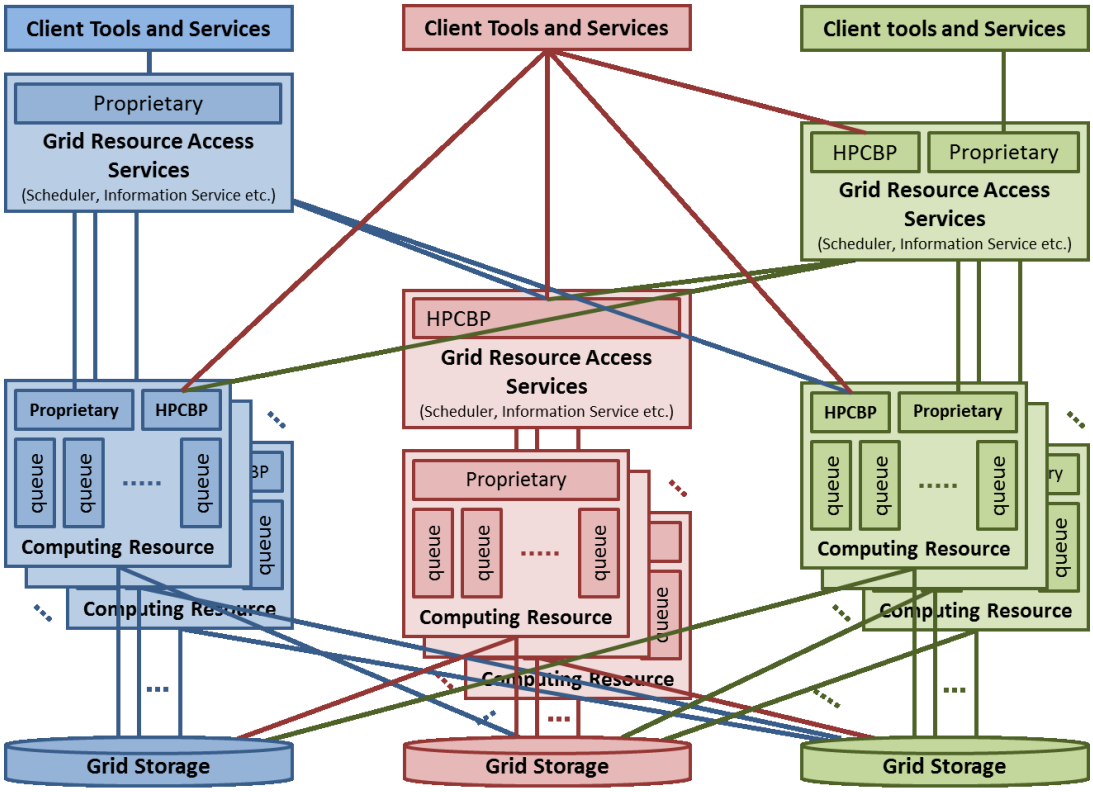


図 1-2　インターオペレーションの概念アーキテクチャ

表 1-1　インターオペレーションアーキテクチャの特徴

Type	インターフェース				特徴
	サービス		コンピュータ資源		
	P	H	P	H	
Type-PH	✓			✓	**長所**：従来のグリッドからの移行工数が最小。**短所**：従来機能のサポートの継続が難しい、後述のロードバランスの乱れ問題が存在。
Type-PD	✓		✓	✓	**長所**：比較的実装が容易で従来機能もサポート可能。 **短所**：後述のロードバランスの乱れ問題が存在。
Type-HH		✓		✓	**長所**：最もシンプルな構成でメンテナンス等の工数が最小。異種グリッドからサービス、コンピュータ資源両方へのジョブ投入が可能である。サービスへのジョブ投入では、後述のロードバランスの乱れ問題が存在しない。 **短所**:従来機能の移行が難しい。後述のメタスケジューラが二重になりジョブ投入レスポンス時間が長くなる問題がある。
Type-HP		✓	✓		特徴は基本的に Type-HH と同じ。**長所：**Type-HH より移行開発工数が少ない。**短所：**異種グリッドからのジョブ投入はサービス経由でしかできない。
Type-HD		✓	✓	✓	特徴は基本的に Type-HH と同じ。**短所：**コンピュータ資源の独自仕様インターフェースにはほとんど意味がない。
Type-DH	✓	✓		✓	特徴は基本的に Type-HH と同じ。**長所：**クライアントとサービス間で従来インターフェースが使用できるため、この間では従来の独自機能への特別な対応は不要になる。
Type-DP	✓	✓	✓		特徴は基本的に Type-HH と同じ。**長所**:HPCBP を異種グリッドからのジョブ投入のみに限定できるため、従来の独自機能に対する特別な対応は必要ない。
Type-DD	✓	✓	✓	✓	特徴は基本的に Type-DP と同じ。異種グリッドからコンピュータ資源への直接ジョブ投入が可能となるが、**長所**として二重にメタスケジューラを通る問題はないが、**短所**としてロードバランスの乱れ問題が発生する。

※　Type-sc の s はグリッド資源アクセスサービス、c はコンピュータ資源のインターフェースを示し、P、H、D はそれぞれ独自仕様、HPCBP 仕様、両仕様に準拠していることを示す。

1.2.3. HPCBP 仕様に準拠した実装

HPCBP 準拠のサービスやモジュールは、多くのグリッドプロジェクトや企業によって開発されている。国立情報学研究所の RENKEI [20]（本研究の HPCBP 実装）、米国 XSEDE

に採用されているバージニア大学の Genesis-II、欧州の EGI で採用されている UNICORE [21]、gLite、ARC [22]、英国のナショナルグリッドで採用されている GridSAM など国策プロジェクトやアカデミックプロジェクトで開発されたものが多い。しかし、なかには Microsoft 社の Windows HPC Server 2008 [23]、バッチスケジューラ LSF の開発元である Platform 社の BES++など一般企業が開発し、製品に組み込んだり、企業発のオープンソースとして公開したりしているものもある。これら実装の開発当初は、図 1-2 の Grid-A のようにコンピュータ資源だけが HPCBP に準拠したサービスを持つアーキテクチャをとっていたが、RENKEI は本研究により、XSEDE、EGI はその更新時に、Gird-C のように資源アクセスサービスで HPCBP を準拠するか、もしくはこれに近いアーキテクチャに変化してきている。RENKEI の実装については第 3 章で、XSEDE、EGI の実装については第 2 章の関連研究にて、その詳細を説明する。

1.3. HPCBP インターオペレーションにおける諸問題

HPCBP 仕様によるインターオペレーションには多くの問題が存在するが、本研究では、準拠すべき仕様が不足している問題、既存のグリッドミドルウェア仕様との整合性の問題を議論する。また HPCBP に限らず、ジョブ実行サービスレベルのインターオペレーションが潜在的に持つ、コンピュータ資源間のロードバランスの乱れによる性能低下問題、グリッド間でのユーザアカウント管理方法の違いに起因するデータステージング回数の増加問題について議論する。本節ではその問題概要を説明し、問題詳細と解決方法の提案は第 3 章と第 4 章で行う。

1.3.1. HPCBP 仕様の問題

- **HPCBP 仕様における機能仕様の欠如**

HPCBP 仕様による異種グリッド間インターオペレーションの最大の問題は、実運用に必要な機能仕様と情報表現仕様が十分には規定されていないことである。実行資源決定段階、投入段階、実行段階など、ジョブの実行や管理の様々な段階における機能仕様、情報表現仕様の多くが不足しており、インターオペレーションの実運用を目指すグリッドミドルウェアを開発できない。具体的には、グリッド間情報交換仕様の欠如、データステージングプロトコル仕様の欠如、認証クレデンシャルのセキュアな委譲仕様の欠如、ユーザアプリケーションのデプロイメント仕様の欠如などが挙げられる。

グリッド間情報交換仕様の欠如は、適切な資源へのジョブ投入ができない、運用上必要なアカウント情報などの交換ができないなど運用上の問題を引き起こし、データステージングプロトコル仕様の欠如は、ジョブは投入できてもデータステージングでエラーが発

生するなどの問題を引き起こす。また、認証クレデンシャルのセキュアな委譲仕様の欠如は、図 1-2 のようなクライアントとコンピュータ資源の間にグリッド資源アクセスサービスが存在する環境では、サービスからコンピュータ資源へ投入するための認証クレデンシャルをクライアントからサービス群へ転送する必要があり、セキュアでない環境になる可能性がある。また、ユーザアプリケーションのデプロイメント機能仕様の欠如はユーザ作成のアプリケーションが実行できない可能性を持つ。

- **従来仕様との整合性の問題**

上記のように、HPCBP はジョブ投入のための最小限の仕様しか規定しておらず、HPCBP サービスとクライアントだけではグリッド環境を構築することはできない。また、既存のグリッドミドルウェアを HPCBP に準拠させる際、ユーザ資産の保護や既存機能の維持などの観点から、既存ミドルウェアの上位互換性を求められる場合が多い。また一般に、HPCBP 仕様よりも独自仕様の方が豊かな機能を持つ。このため、従来のサービスやモジュールを残しつつ、HPCBP 仕様のサービスやモジュールを追加することが一般的である。このとき問題となるのが、既存仕様と HPCBP 仕様の整合性の問題である。例えば認証仕様の違いである。認証クレデンシャルや認証方法は、各グリッドシステムの基本であり、ミドルウェアアーキテクチャは各グリッドのセキュリティポリシーを基に決定され、全てのサービスやモジュールで認証やセキュリティ上の問題が発生しないように注意深く設計される。このため、認証クレデンシャルや認証方式が異なると、基本アーキテクチャ設計時のセキュリティポリシーが順守できず、基本アーキテクチャの変更が必要になる場合があり、非常に大きな問題となることがある。

1.3.2. 性能問題

- **計算機資源のロードバランスの乱れ問題**

現在、HPCBP に準拠しているミドルウェアは、図 1-2 の Grid-A のようにコンピュータ資源のみに HPCBP サービスを実装しているものが多い。このようなアーキテクチャで、Grid-A のコンピュータ資源に直接他のグリッドからジョブが投入される場合を考える。このとき、ジョブ投入側のグリッドのアーキテクチャは問わない。Grid-A のユーザが Grid-A のコンピュータ資源にジョブを投入する場合は、Grid-A のメタスケジューラが配下のコンピュータ資源のロードバランスをとるため効率的なジョブ投入ができる。しかし、ここで他のグリッドが Grid-A のコンピュータ資源にジョブを直接投入すると、Grid-A のメタスケジューラを経由しないので、メタスケジューラが考慮したコンピュータ資源間のロードバランスが乱れ、ジョブの効率的な実行に影響を与えることが考えられる。これは、グリッ

ド間の情報交換仕様の欠如も原因となっているが、実際には、単なる情報交換だけでは解決できない。インターオペレーションの実運用には、解決しなくてはならない問題である。

- **データステージング回数の増加問題**

グリッド環境のコンピュータ資源のユーザアカウントの管理方法は二種類ある。従来の単体のコンピュータシステムと同様な静的ユーザアカウント管理と、ユーザアカウントがジョブ毎に動的に割当てられる一時ユーザアカウント管理である。また、アカウントの管理方法の違いのため、ジョブの作業領域の管理方法も異なる。静的アカウント管理では、単体システム同様な持続的なユーザ毎の home 領域とジョブ毎の一時作業領域の両方を持つことが可能である。ユーザアカウントと作業領域が恒久的なため、ジョブの入力ファイルや結果ファイルはジョブの実行が終了しても、消去しない限りいつでもアクセス可能である。しかし、一時アカウント管理の場合は、一般に恒久的なストレージ領域を持つことはできない。一時アカウントと一時作業領域のライフタイムは、ジョブ実行中に限られるため、ジョブへの入力ファイルと出力ファイルはジョブ内で外部資源との間で転送しなくならない。このため、依存性のある二つのジョブを実行する場合には、先行ジョブの結果をジョブ内で外部ストレージ資源に転送し、後行のジョブはジョブ内で外部ストレージ資源から先行ジョブの結果を転送してから実行するといった二回のデータ転送が必要になっている。一方静的アカウント管理のグリッド環境では、上記のようにユーザアカウントと home のライフタイムは持続的なので、コンピュータ資源間で直接にデータ転送が可能である。インターオペレーション環境で一時アカウント管理のコンピュータ資源と静的アカウント管理のコンピュータ資源とで依存性のあるジョブを実行する場合には、一時アカウント管理資源のユーザアカウントと作業領域のライフタイムの制限のため、静的アカウント管理のコンピュータ資源も、一時アカウント管理に準じた間接的なデータ転送を行わざるを得ない。計算機の高性能化とデータの巨大化が進む一方である現代では、ジョブの実行時間におけるデータ転送時間の比率は大きくなる一方であり、同一データを二回転送することはジョブの長期化、ネットワーク負荷の増大をもたらすため、解決すべき問題である。

1.4.　本研究の目的と貢献

本研究の目的は大きく二つある。第一の目的は、実運用を考慮したインターオペレーションアーキテクチャを提案し、実際にミドルウェアを開発して他グリッドと実証実験を実施し、その有効性を示すことである。具体的には、まず単一グリッドでのジョブ実行動作を基に、インターオペレーションの各ジョブ実行動作ステップで必要とされる機能や情報を明確に示す。その上で、これら必要な機能、情報に対する HPCBP 仕様の問題点を明

らかにし、解決方法とアーキテクチャを提案する。この提案を基に NAREGI ミドルウェアをベースにした HPCBP 仕様準拠ミドルウェアを研究開発する。さらに、これを使用してインターオペレーション研究のための環境を構築し、他グリッドミドルウェアとのインターオペレーション実証実験を行う。第二の目的は、上記のインターオペレーション環境で発生が予想される問題に対して、解決方法の提案と有効性の確認である。具体的には、計算機資源間のロードバランスの乱れによる性能低下問題と、ユーザアカウント管理方法の違いによるデータステージング回数問題について、実験とシミュレーションにより発生することを確認し、解決方法を提案する。さらに実験やシミュレーションによって提案方法の有効性を確認する。

1.5.　本論文の構成

本節では本章に引き続く章の概要を説明する。

第 2 章では、関連研究として、インターオペレーション関連の標準仕様、インターオペレーションの実装例、グリッド以外の環境との資源連携、複数のメタスケジューラによる共有資源のスケジューリングについて紹介する。標準仕様の紹介では、HPCBP 仕様自体と HPCBP が準拠すべき仕様として定義している、ジョブ記述仕様 Job Submission Description Language（JSDL）、ジョブ実行管理仕様 Basic Execution Service（BES）、認証クレデンシャル仕様である X.509 証明書とユーザ名／パスワードについて概要を説明する。また、OGF で策定中の、実運用可能なレベルの次世代インターオペレーション仕様 PGI についても検討状況を説明し、さらにグリッド環境に対する API 仕様である Simple API for Grid Applications（SAGA）によるインターオペレーションの概要を説明する。SAGA は HPCBP や PGI とは異なり API であるためアプリケーションから直接グリッド資源をアクセスすることができる。次に HPCBP 準拠の実装例として、米国の XSEDE、欧州の European Middleware Initiative（EMI）のアーキテクチャを説明する。両グリッド環境とも、ローカルグリッド環境の機能として HPCBP 仕様には定義されていない様々な機能を持っているが、他のグリッドとのインターオペレーションのため、HPCBP 仕様のジョブインターオペレーションも可能である。また、HPCBP 以外の仕様によるインターオペレーションとして、NAREGI と欧州の gLite 間、欧州のグリッドとインドのグリッド間のインターオペレーション、SAGA によるインターオペレーション実験を紹介する。さらに、グリッド環境とグリッド以外の環境との資源連携として、実験装置に接続されているコンピュータやユーザが身近に所有するローカルシステムとグリッドの計算資源の連携について紹介する。また、クラウド環境をグリッドの資源として使用する資源連携についても説明する。さらに、複数のメタスケジューラが資源を共有するために発生するロードバランス問題の一解決手段として、WS-Agreement 仕様に準拠する方法を説明する。

第 3 章では、単一グリッド環境のジョブ投入ステップで必要となる機能、情報などの説明を行い、これをインターオペレーション環境として拡張するときに必要となる機能や情報を明らかにする。そして、この必要機能や情報を満たすインターオペレーションアーキテクチャの提案を行なう。また、現実には HPCBP 仕様に基づくインターオペレーションを行う必要があり、HPCBP で不足している機能や情報を明確にし、解決策を提案する。さらに、既存のミドルウェアに HPCBP 準拠機能を追加するときの問題点を、NAREGI への HPCBP 準拠機能追加を例に明らかにし、解決策を提案する。この解決策を基に実装を行う。さらに、この実装を使用して今後のインターオペレーション研究のための基盤環境を構築し、諸外国のグリッドとのインターオペレーション実証実験を実施する。最後にその結果を報告する。

第 4 章では、コンピュータ資源間のロードバランスの乱れによるジョブ実行時間の長期化問題と、一時ユーザアカウント管理のコンピュータ資源間や一時ユーザアカウント管理と静的アカウント管理のコンピュータ資源間でのデータステージング回数問題について解決策の提案を行う。ロードバランスの乱れについては、ローカルグリッド環境のメタスケジューラがコンピュータ資源のロードバランスを考慮したジョブ配置をしても、他グリッドからのコンピュータ資源への直接ジョブ投入により発生することが予想されることを説明し、実験とシミュレーションにより本問題が発生することを示す。このときのメタスケジューラ、情報サービス、コンピュータ資源の動作を検討し、原因は他グリッドからのジョブ投入をメタスケジューラが検出するのが遅れるためであることを示す。解決策として、他グリッドからのジョブを一括して受信し、さらにメタスケジューラとスケジューリング情報を共有するゲートウェイを設けることを提案する。このゲートウェイは、他グリッドからのジョブをコンピュータ資源にフォワードする前に、メタスケジューラと共有しているスケジューリング情報を更新する。これにより、メタスケジューラはコンピュータ資源に他グリッドからのジョブが投入される前に、スケジューリング情報からこれを知ることができる。一時ユーザアカウント管理のコンピュータ資源間や一時ユーザアカウント管理と静的アカウント管理のコンピュータ資源間でのデータステージング回数問題については、一時ユーザアカウント管理のコンピュータ資源では、ユーザアカウントと作業領域のライフタイムがジョブ実行中に限られるため、ジョブ間でのデータステージングは一時ストレージを経由する必要があることを示し、ジョブ間で直接データステージングができる静的アカウント環境に比べデータステージング回数を 1 回多く必要とすることを示す。この 1 回多いデータステージングは、データが巨大化しステージング時間に長時間を要する近年、ジョブ実行時間の長期化、ネットワークトラフィックの増大の原因となり問題である。そこで一時アカウント環境のコンピュータ資源とでも直接データ転送する方法を提案する。具体的には、ジョブドキュメントによるジョブの HOLD 機能、オペレーションによる HOLD 解除機能、これらに対応するメタスケジューラの機能を提案する。これを現在 PGI の機能

要件に照らし合わせて実装し、実験で有効性を確認したことを示す。

第 5 章では、本論文のまとめと将来構想として、ジョブ投入・実行インターオペレーションだけでなく、情報交換インターオペレーションなどを実現し、単一グリッドと比べ遜色のないレベルの機能を持つインターオペレーションアーキテクチャの実現や、ローカルシステムやクラウド環境との連携を目指すべきであることを示す。

第 2 章　　関連研究

2.1.　標準仕様

- **HPC Basic Profile（HPCBP）**

第 1 章で述べたように、HPCBP は機能仕様ではなく、インターオペレーションを実現するために準拠すべき仕様を規定するプロファイル仕様である。HPCBP では、ジョブ記述仕様として OGF の Job Submission Description Language（JSDL） [24]と JSDL の HPC Profile Application Extension 仕様 [25]を、ジョブのスケジューリングと管理仕様として OGF の Basic Execution Service（BES） [26]を、セキュリティ（認証クレデンシャル）仕様として IETF の X.509 証明書 [27]もしくはユーザ名／パスワードトークン [28]を規定している。さらに、これら仕様で規定される様々な機能や情報表現のなかで、必ず準拠すべき項目や制限について規定している。

- **Job Submission Description Language（JSDL）**

JSDL は、主としてバッチジョブのための XML 形式のジョブ記述言語である。単一ジョブの実行要件の記述を目的に規定しており、ワークフロージョブや依存性のあるジョブは JSDL だけでは記述できない。また、ジョブを実行するキュー名など、ジョブ実行時に決定されるような情報も記述できない。これらの情報はジョブドキュメント内の JSDL 以外の部分で記述されることを期待している。一般にジョブワークフローやジョブ間の依存性の

制御はシステムにより異なるため、このようにジョブ要件のみの記述する仕様とすることで、JSDL ドキュメントのポータビリティを高くしている。システムへの依存性の少ない記述は、インターオペレーション環境で任意のグリッドに投入するジョブを記述するには適している。

JSDL は、*Job Structure*、*Job Identification*、*Application*、*Resources*、*Data Staging* の 5 つのコアエレメントから構成されている。*Job Structure* エレメントはその他 4 つのコアエレメントを内包して 1 つのジョブとして形作る最外殻のエレメントである。*Job Identification* エレメントではジョブを識別するための命名や説明を記述し、*Application* エレメントでは実行するアプリケーションのパスなどの記述し、*Resources* エレメントではこのジョブを実行させるのに必要な資源条件を記述し、*Data Staging* エレメントでの入力ファイルや結果ファイルなど外部資源とデータ転送を行う必要があるファイルの情報を記述する。HPCBP 仕様では、*JobIdentification* エレメントの *JobName*、*JobProject* サブエレメントを、*Application* エレメントの HPC Profile Application Extension 仕様の *HPCProfileApplication* サブエレメントを、*Resources* エレメントの *CandidateHosts*、*ExclusiveExecution*、*OperatingSystem*、*CPUArchitecture*、*TotalCPUCount* サブエレメントのサポートを必須としている。ただし、ジョブ実行サービスとしてサポートが必須というだけで、ユーザが記述する JSDL ドキュメントには記述されていなくても良い。なお、ユーザが記述した JSDL ドキュメントは、ジョブ実行過程においてグリッドサービス間を伝搬していく。このとき、グリッドサービスが決定した内容を次のグリッドサービスに伝達するためにも使用され、サービスによってエレメントの追加記述や内容の変更が行われる。

JSDL には *FileSystem* や *CandidateHosts* など一見実行環境や実行時に依存しているように思えるエレメントも存在する。しかし、*FileSystem* エレメントは ”HOME”、”ROOT”、”SCRATCH” など、一般的にシステムが持つファイルシステムを抽象化した表現で記述し、実行環境依存にはなっていない。ただし、*CandidateHosts* は例外的である。*CandidateHosts* には具体的なホスト名を記述する。ユーザが記述することも可能であるが、一般的には資源ブローカが使用可能な資源を絞り込み JSDL ドキュメントに追加してメタスケジューラに伝達するときなど、サービス間の伝達のために使用する。このため、ユーザが記述しなければ実行環境に非依存である。しかし、資源ブローカが存在しない環境では、ユーザが実行する資源を指定せざるを得ず、実行環境依存となる。

- **Basic Execution Service（BES）**

BES は OGF 標準のアーキテクチャ Open Grid Service Architecture（OGSA）に沿った Web サービス形式のジョブ実行サービスである。既存のグリッドミドルウェアのジョブ実

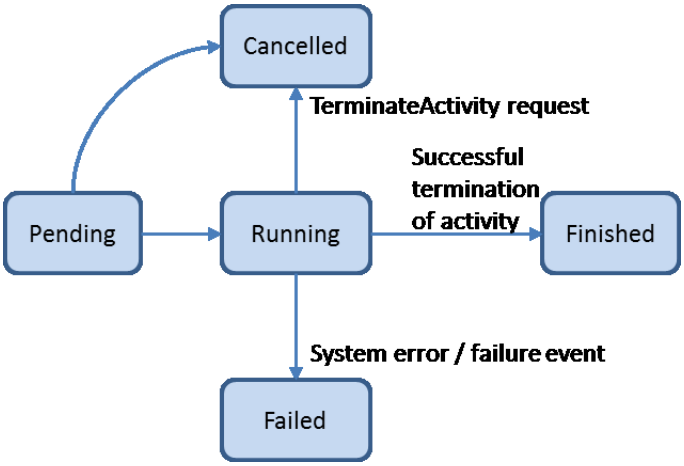
行サービスには各グリッド固有の特殊な機能を持つ場合が多いが、BES はジョブ実行管理に必要な基本的なジョブのステートモデルとオペレーションしか持たない。BES のオペレーションには、BES 自身を管理するオペレーションを定義した BES-Management port-type と、アクティビティの生成、モニタ、管理などのオペレーションを定義した BES-Factory port-type に区分される。この他に生成した個々のアクティビティのモニタや管理機能の拡張用に BES-Activity port-type もあるが、オペレーションは定義されていない。BES-Management port-type と BES-Factory port-type のオペレーションの一覧をそれぞれ表 2-1、表 2-2 に、また BES のジョブステートモデルを図 2-1 に示す。

表 2-1　BES-Management port-type

オペレーション	概要
StopAcceptingNewActivities	BES に対する新しい Activity の受け取り停止指示
StartAcceptingNewActivities	BES に対する新しい Activity の受け取り開始指示

表 2-2　BES-Factory port-type

オペレーション	概要
CreateActivity	引数の ActivityDocument にしたがいアクティビティを生成する。一度に生成できるアクティビティは 1 つのみである。正常に生成されるとアクティビティ識別子として End Point Reference（EPR）が戻る。
GetActivityStatuses	引数（EPR）で指定したアクティビティのステータスを取得する。一度に複数のアクティビティのステータスを取得可能で、このときは複数の EPR を指定する。
TerminateActivities	引数（EPR）で指定したアクティビティを終了させる。実行中であればキャンセルする。一度に複数のアクティビティをキャンセルすることが可能で、このときは複数の EPR を指定する。
GetActivityDocuments	引数（EPR）で指定したアクティビティの ActivityDocument を取得する。この ActivityDocument は CreateActivity で投入された ActivityDocument に対し実行時に決定された情報で変更されたものになっている場合がある。一度に複数の ActivityDocument を取得可能で、このときは複数の EPR を指定する。
GetFactoryAttributesDocument	BES-Management アトリビュートを記述した XML ドキュメントを取得する。BES-Management には資源情報などが含まれている。



（出典： “HPC Basic Profile”, [15]）

図 2-1　BES のジョブステートモデル

BES のジョブステートモデルは図に示すステートをメインステートとし、各ステートの中にサブステートを持つ拡張を許している。サブステートは実装依存のため、拡張したサブステートをサポートしていないクライアントやサービスでは無視される。このため、インターオペレーションでサブステートを使用するには、プロファイル仕様での明確な定義が必要となるが、HPCBP では定義していない。

- **X.509 証明書**

公開鍵暗号方式の認証クレデンシャルである。個人、組織、サーバの公開鍵の正当性を保証する信用のおける第三者機関である認証局が発行する公開鍵証明書である。公開鍵証明書には所有者情報と公開鍵の対が認証局の秘密鍵でデジタル署名されている。認証局による署名を検証することによって、公開鍵の正当性を確認することができる。

- **Production Grid Infrastructure（PGI）**

BES の項で述べたように、BES はジョブ実行管理に必要な基本的なジョブのステートモデルとオペレーションしか持たない。このため、実際に運用レベルのインターオペレーションを行うためには、グリッド間の情報交換仕様、認証クレデンシャルのデレゲーション仕様などさらに多くの仕様への準拠や機能の追加を規定する必要がある。この不足仕様や機能については「3.2. インターオペレーション仕様とアーキテクチャの提案」で議論する。

PGI 仕様は HPCBP 仕様と同様にプロファイル仕様であるため、PGI-WG [29]は実運用におけるユースケースなどから機能要件や情報表現要件を抽出し、BES や JSDL などの基本仕様の WG にこれら要件の追加を依頼する形態で議論を進めている。当初、2009 年頃には仕様策定を終了する予定であったが大幅に遅延しており、2012 年 6 月現在、BES、JSDL、GLUE の各 WG が PGI-WG の要件を基に検討を継続中である。従来の BES の基本的ジョブステートとオペレーションと異なり、PGI 要件ではデータステージングをジョブ実行とは別に行うことを可能にし、ジョブ実行時のデータステージング待ちによる資源の遊休時間をなくすことができるなど、実運用上の機能を意識した要件となっている。本研究では、データステージング回数問題の解決策に、この要件の一部を使っている。

- **Simple API for Grid Applications（SAGA） [30]**

HPCBP や PGI は Web サービスとして実装されるジョブ実行サービスの仕様であり、アプリケーションの実行は、各グリッドのクライアントツールから実行要件を記述したジョブドキュメント投入することにより行われる。これに対し、SAGA はグリッド資源を利用するための API であり、アプリケーションプログラムから直接グリッド資源を利用することができる。この API を使用すれば、特定のグリッド環境に依存しないインターフェースで各グリッドにアクセスできるため、ポータブルなプログラミングが可能となる。SAGA ではコンピュータ資源だけでなく、グリッドストレージ資源へのアクセスインターフェースやサービス検索インターフェースも定義しており、HPCBP に比べ実用的なものになっている。大規模運用グリッドである European Middleware Initiative（EMI） [31]や XSEDE でも採用されている。SAGA のサンプル実装には、抽象化されたグリッドアクセスアダプタインターフェースがあり、アダプタでこの抽象インターフェースを各グリッドの具象インターフェースに変換する。様々なグリッドに対するアダプタを用意することで、ひとつのアプリからで様々なグリッド環境へアクセスすることが可能になっている。ただし、認証クレデンシャルに対する操作は抽象化されておらず、例えば X.509 プロキシ証明書[32]を使う場合には、myproxy-init などの操作を API を使用する前に行う必要がある。

2.2. インターオペレーション実装

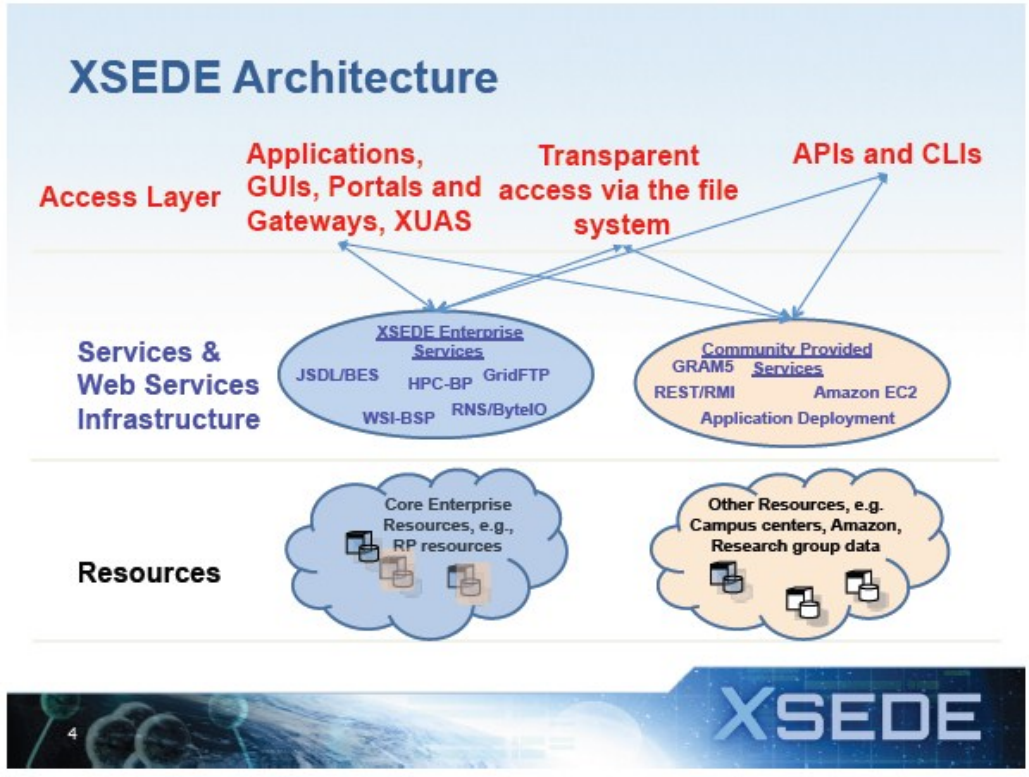
2.2.1. HPCBP 仕様によるインターオペレーション

- **XSEDE**

グリッドコンピューティングの初期から有名であった米国の TeraGrid は 2011 年にサービスを終え、新たなアーキテクチャを持つ XSEDE に引き継がれた。これまでの TeraGrid は、独自仕様のグリッドツールキットである globus を中心に構成され、グリッドとして閉

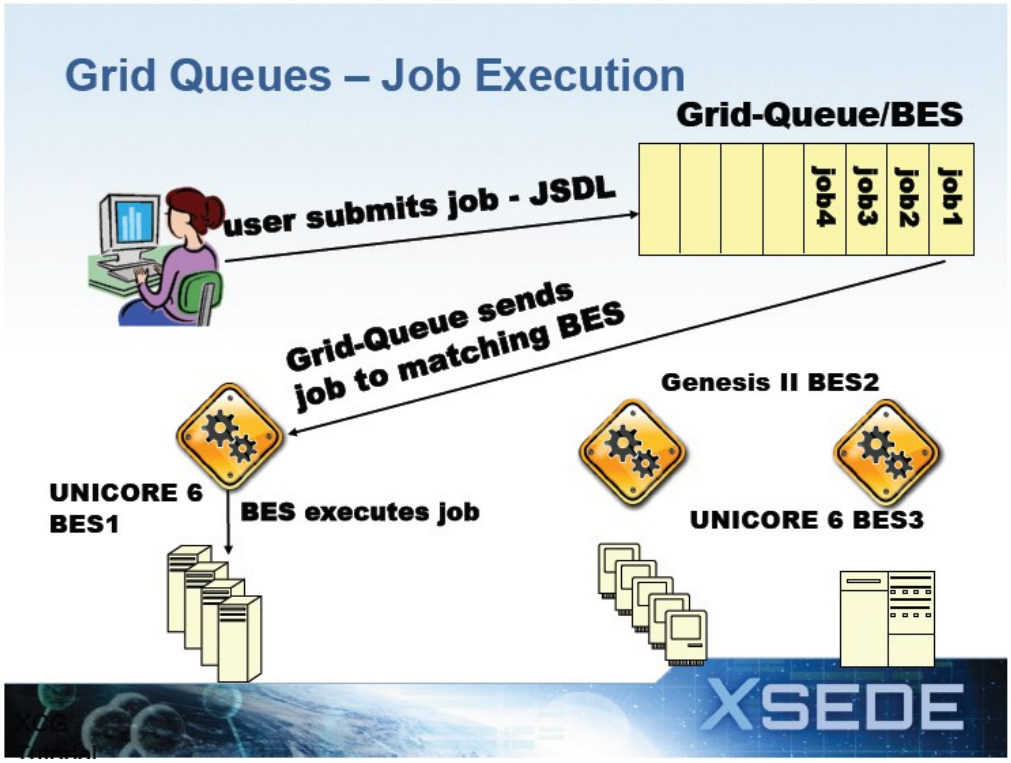
じていた。これに対し、XSEDE では標準仕様のインターフェースに準拠しており、大学などのキャンパスグリッド環境や他グリッド環境との連携を重視していることが特徴である。また、従来の globus [33]サービスや新しいクラウド技術などとの親和性も考慮し、これらもコミュニティサービスとしてサポートしている。ミドルウェアは 3 種類あり、環境に応じて選択してインストールすることができる。まず大型センターなどで使われる XSEDE Enterprise 環境には欧州で開発された UNICORE6 が、Community 環境として globus が、キャンパスグリッド環境としてバージニア大学が開発した Genesis-II が用意されている。

XSEDE の基本アーキテクチャを図 2-2 に示す。この図からは一見、図 1-2 の概念アーキテクチャの Grid の構成図のいずれにも属さないように見える。しかし、図 2-3 によると Grid-Queue/BES はメタスケジューラの一種と考えることもできるため、Grid-A または Grid-C のタイプの構成と思われる。クライアントと Grid-Queue/BES 間のインターフェース仕様が不明のため、どちらのタイプかは断定できない。



（出典： “XSEDE Architecture Overview & Context”, [34]）

図 2-2　XSEDE の基本アーキテクチャ



（出典： “XSEDE Architecture Overview & Context”, [34]）

図 2-3　Grid Queue

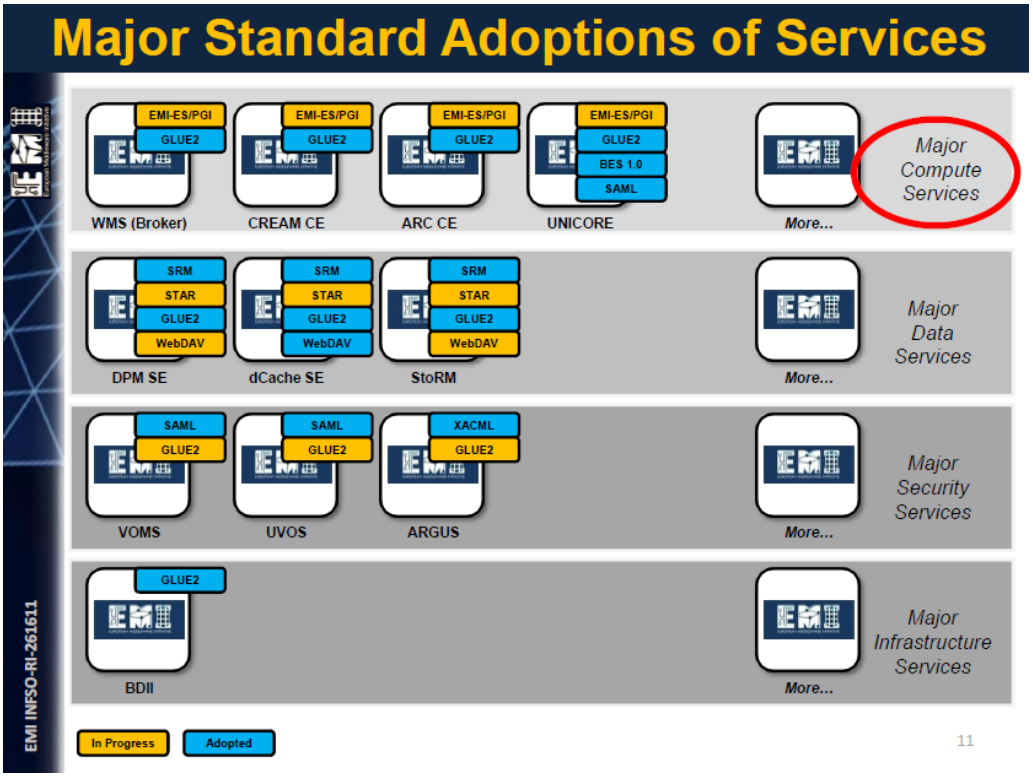
- **European Middleware Initiative（EMI）**

EU は 2010 年に欧州内のグリッドコンピューティング環境、ミドルウェアの開発管理の大幅な組織改定を行った。従来独立して運用されていた DEISA、EGEE、NORDUGRID、D-Grid と各国のナショナルグリッドが、HPC 向けの Partnership for Advanced Computing in Europe （PRACE） [35]と HTC 向けの EGI に再編された。これにより、EGI は 2011 年 5 月現在、参加国数: 51（アジア、北米、南米を含む）、参加機関数: 338、CPU core 数: 240,000、ディスク容量: 102PB、テープアーカイブ容量: 89PB まで膨れ上がり、一万人以上のユーザが一日に数十万ジョブを実行している。この再編に伴い、複数の組織で開発、管理されていたグリッドミドルウェア群も EMI に統合された。この EMI への統合に伴い、管理下のグリッドミドルウェア間のインターオペレーションが求められ、UNICORE、gLite、ARC 間のインターオペレーション機能が開発された。これら 3 つのグリッドミドルウェアは HPCBP プロトタイプを開発していたが、図 2-4 によると現在 HPCBP（BES）をサポートしているのは UNICORE と gLite のみである。EMI は、実運用に耐えうるより高機能なインターオペレーション用ジョブ実行サービス仕様 EMI Execution Service （EMI-ES） [36]を独自に策定し、最終的にこれら 3 つのミドルウェア

は EMI-ES によってインターオペレーションを実現する予定である。EMI-ES はヨーロッパの国際標準と言えるが、将来的には PGI にも準拠する予定がある [37]。図 2-5 に EMI のジョブ実行の流れを示す。これにより UNICORE、ARC は図 1-2 の Grid-A に近い構造を、gLite は Grid-C の構造を持っていることが分かる。

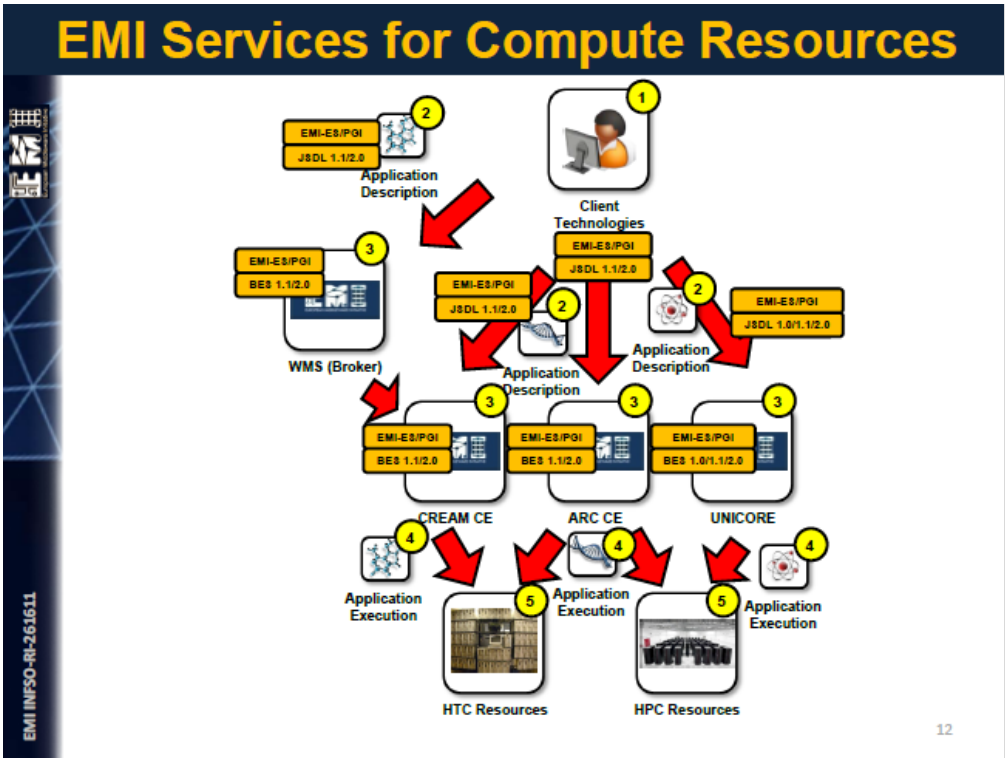
- **その他**

その他､英国の GridSAM などいくつものプロジェクトが HPCBP をサポートしている。多くはアカデミックプロジェクトであるが、製品や製品関連のオープンソースソフトウェアで実装されている例もある。米国マイクロソフト社は、Windows HPC Server 2008 にグリッド環境からのジョブ投入に対応するために HPCBP インターフェースを実装しており、また米国 Platform 社も同社のバッチスケジューラ製品である LSF とオープンソースとして公開しているバッチスケジューラ Lava 用に BES クライアント／サーバ BES++をオープンソースとして公開している。



（出典： “EMI Standard Adoptions”, [37]）

図 2-4　EMI のモジュール構成



（出典： “EMI Standard Adoptions”, [37]）

図 2-5　EMI のジョブ投入の流れ

2.2.2. HPCBP 仕様以外のインターオペレーション

- **NAREGI-gLite**

第 1.1 節のインターオペレーションの経緯で説明したように、インターオペレーションは少数のグリッドプロジェクト間で、独自仕様で可能な部分から始めてアイランドを構成することから始まった。その一環として実施されたのが、日本の国立情報学研究所の NAREGI と欧州 EGEE プロジェクトの gLite の間でのインターオペレーション研究開発である [38]。HPCBP によるインターオペレーションと同様にジョブ実行サービスレベルのインターオペレーションであるが、サービスインターフェースは独自仕様である。独自仕様での実現の難しさから、NAREGI から gLite へのジョブ投入とその逆とで構成が非対称になっている。アーキテクチャを図 2-6 に示す。NAREGI→gLite では NAREGI-Portal（クライアント）→NAREGI-SS（メタスケジューラ）→LcgCE（コンピュータ資源）のシンプルな流れであるのに対し、gLite→NAREGI では gLite-UI（クライアント）→gLite-WMS（メタスケジューラ）→gLiteCE（コンピュータ資源）→NAREGI-SS（メタスケジューラ）→

NAREGI-GridVM（コンピュータ資源）と複雑な流れになっている。これは、gLite-WMS を Grid-VM に対応させることが難しいことから、このような構造とした。

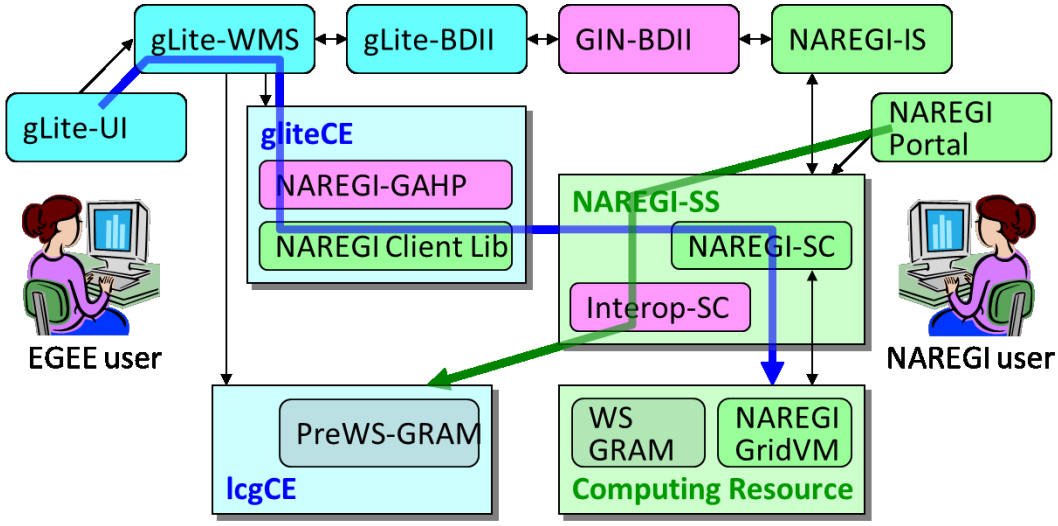
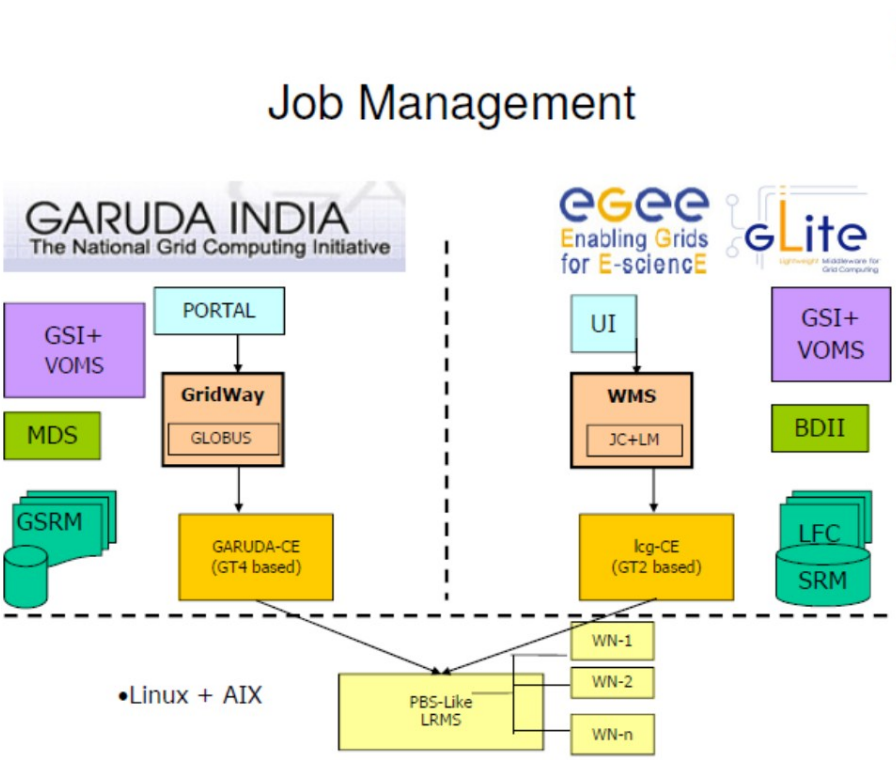


図 2-6　NAREGI-gLite インターオペレーションのアーキテクチャ

- **EU-India**

EU は、欧州のグリッド環境と様々な国のグリッド環境とのインターオペレーションプロジェクトを実施している。例えば EU-China、EU-India、EU-Asia などである。これらの国々のグリッド環境とは独自仕様でインターオペレーションを実現することがほとんどである。そのひとつが EU-India グリッドである [39]。このプロジェクトでは、Information System、Job Management、Data Management、Security、Network のインターオペレーションをテーマとし、ジョブを相互に実行するために必要な項目は網羅している。しかしジョブインターオペレーションは、グリッドレベルではなくバッチスケジューラレベルで実現している。アーキテクチャを図 2-7 に示す。これをインターオペレーションと呼べるかは疑問が残るが、インドのグリッドである Garuda と欧州のグリッドである gLite で同じコンピュータ資源にジョブを投入することは可能である。

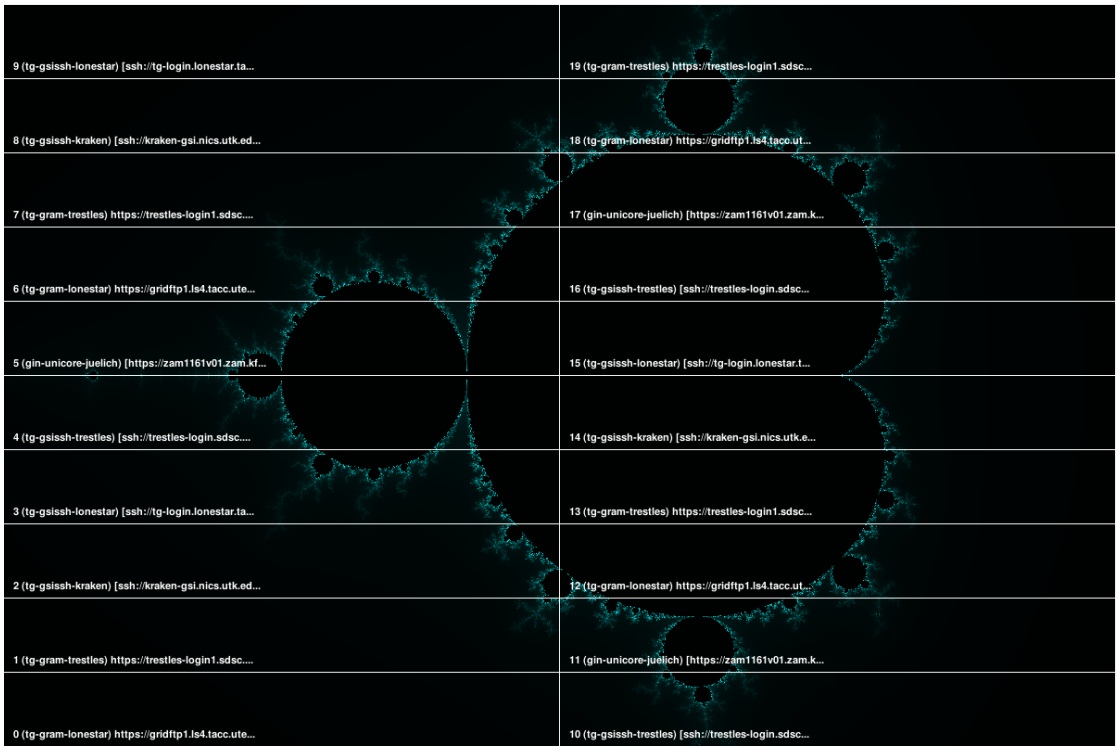


（出典：”Interoperability challenges in Garuda GTS and EGEE grids”, [40]）

図 2-7　EU-India インターオペレーション

- **SAGA によるインターオペレーション**

SAGA には C++のコアライブラリ、Python 実装、Java 実装が開発され、参照実装として公開されている。上記のように C++コアライブラリは様々なミドルウェアと接続するアダプタを持つ。ジョブ用アダプタの安定版として SSH、Globus、Condor、LSF、PBS(Pro)、Torque、BES などが、実験版して UNICORE、NAREGI、EC2 [41]、GridSAM、gLite CREAM、DRMAA などのアダプタなどが公開されている。またファイル用アダプタとして SSH、Globus（GridFTP）などが、コンテキスト用アダプタとして X.509 アダプタが公開されている。参照実装のデモのひとつである、マンデルブロー曲線を複数のグリッドのコンピュータ資源で分割して描画した結果を図 2-8 に示す。



（出典： Open Grid Forum SAGA セッションにおけるデモンストレーションより）

図 2-8　SAGA のデモ

2.2.3. 異なる環境との資源連携

- **ローカルシステム・グリッド間連携**

コンピュータ資源の連携に関する障壁は、異なるグリッド環境間だけではない。ユーザのローカルな計算機システムや実験装置に接続された計算機システムと、これらで生成されたデータを処理するグリッド環境との連携も問題になっている。一般的な科学計算は、データ収集や生成を含む前処理、目的の計算である本処理、計算結果の可視化などの後処理のワークフロージョブであることが多い。データは実験装置や計測装置によって収集されローカルな計算機システムで前処理されるのが一般的である。この前処理されたデータは、目的の計算のためにグリッド環境のストレージ資源に転送され、そのグリッド環境のコンピュータ資源により大規模な解析やシミュレーションなど目的の処理が実行される。この結果は再びユーザの手元のローカルな計算機システムに転送され、可視化などの後処理が実行される。従来のグリッド環境では、グリッド環境内のストレージ資源やコンピュータ資源しか制御できないため、ローカルシステムによる前処理とグリッドコンピュータ資源による本処理の間、グリッドコンピュータ資源の本処理とローカルシステムによる後処理の間に不連続が生じ、ユーザによるインタラクティブな操作が必要になっていた（図 2-9

参照)。このため、夜間や週末などにこのインタラクティブ操作が必要になった場合、ユーザが気づくまで処理が先に進まず無駄な時間が発生するという問題があった。多くのグリッドプロジェクトがこの問題の解決を試みており、国立情報学研究所を中心とした RENKEI プロジェクトでも、ユーザのローカル環境と NAREGI グリッド環境を連携させる研究が実施された（2008 年 9 月~2012 年 3 月)。

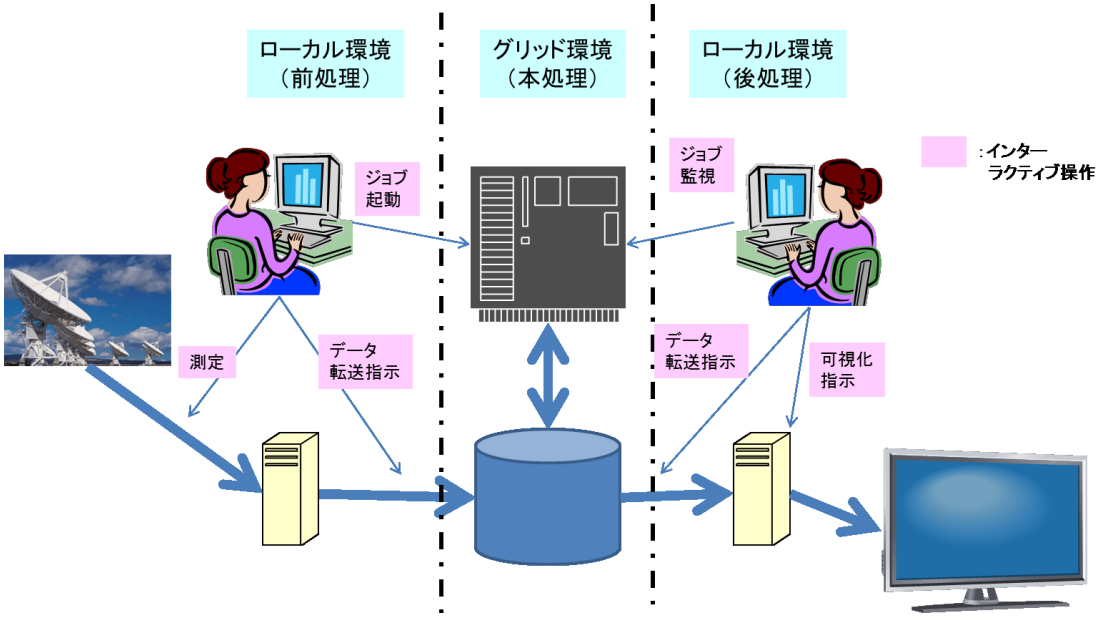


図 2-9　ローカルシステム・グリッド間連携

● **グリッド・クラウド間連携**

近年、コンピュータ資源の所有・運用コストの低減や不足資源の補完のために、クラウドコンピューティングという新たな概念が考案され、大規模な商用環境も多数構築されている。最近更新された XSEDE や EGI でもその機能が一部取り込まれるか、検討されている。クラウドコンピューティングでは、計算環境を仮想マシンとして提供する IaaS から、アプリケーションをサービスとして提供する SaaS までいくつかの形態が考えられている。現在のところ、アカデミックな科学計算環境としては主に IaaS が実験的に利用されているが、アプリケーションをサービスとして提供する商用 SaaS 環境も存在している。アカデミックな科学計算ユーザは独自にアプリケーションを開発する場合が多く、この分野では今後も IaaS が使われることが多いと思われる。クラウドコンピューティングとグリッドコンピューティングは、他組織が所有するコンピュータ資源の利用を可能とする手段のため混同されがちであるが、IaaS ではコンピュータ資源の提供単位が仮想マシンであり、グリッドコンピューティングでのコンピュータ資源の提供単位は計算プロセスであるという

点に大きな違いがある。グリッド・クラウド連携では、グリッド環境で不足した資源をクラウド環境上にグリッドコンピュータ資源としての仮想マシンを構築するといった使われ方をする（図 2-10 参照）。

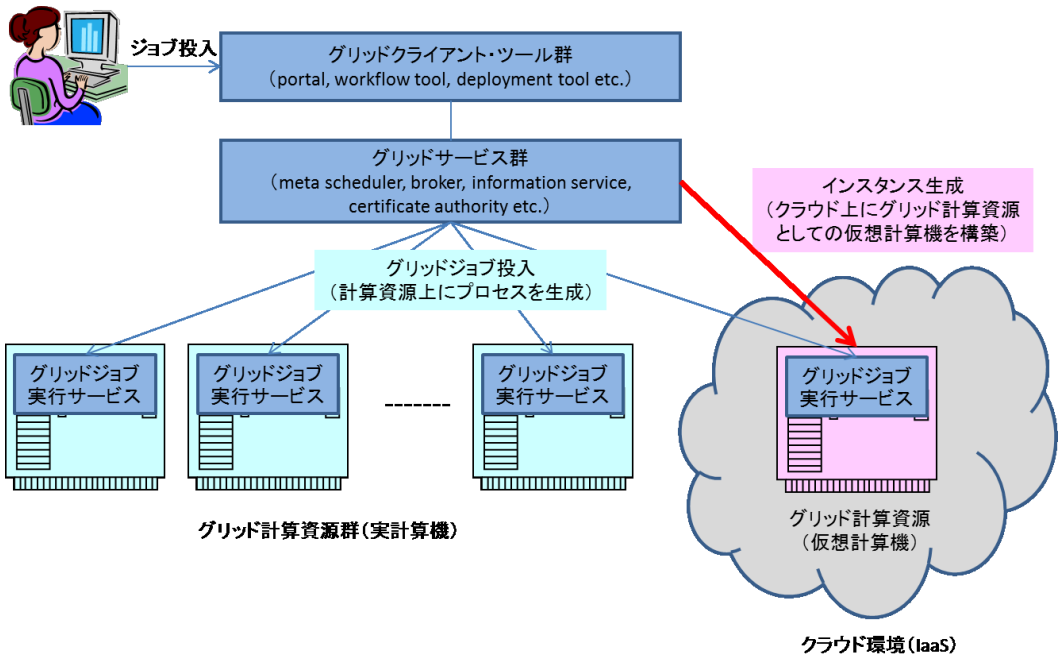


図 2-10　グリッド・クラウド間連携

2.3.　複数のメタスケジューラによる共通資源のスケジューリング

第 1 章で説明したインターオペレーション時のロードバランスの問題は、複数の独立した相互に情報交換を行わない管理主体（本研究の場合はメタスケジューラ）が、不十分な資源情報を基に、各主体が独自にロードバランスをとりながら、共通のコンピュータ資源にジョブを投入するために発生する。このような問題を議論した関連研究は見いだすことはできなかった。しかし、本研究のロードバランスの乱れ問題は、メタスケジューラ間でネゴシエーションを行うことで解決できると考えられる。例えば、WS-Agreement [42]仕様を基にしたジョブ投入ネゴシエーションを行うことで解決できると考えられる。しかし、HPCBP 仕様にはネゴシエーション仕様はなく、また WS-Agreement 仕様でネゴシエーションをとるとしても、そのネゴシエーションの標準仕様を決める必要がある。これには多くの工数を必要とする。またこれをすべてのグリッドミドルウェアに実装するには労力が大きく時間も要する。標準インターフェースの仕様は少ないほど各ミドルウェアが対応しやすく、インターフェースの実装上の問題も発生しにくい。

2.4.　本章のまとめ

本章では、本研究のテーマである異種グリッド間インターオペレーションについての関連研究、関連資料の調査を行った。まず、インターオペレーション関連の標準仕様としてインターオペレーション仕様 HPCBP、ジョブ記述言語仕様 JSDL、ジョブ実行サービス仕様 BES、認証クレデンシャル仕様 X.509 証明書、次世代インターオペレーション仕様 PGI、グリッドアプリケーション API 仕様 SAGA の概要説明を行った。次に標準仕様を基にしたインターオペレーション機能の実装研究として、米国の XSEDE、欧州の EMI の概要を、独自仕様によるインターオペレーション実装研究として、NAREGI-gLite、EU-India の概要を説明し、さらに SAGA によるインターオペレーション研究の概要を説明した。次にグリッドとグリッド以外の連携研究として、ローカルシステム・グリッド連携、グリッド・クラウド連携の概要を紹介した。以上の関連研究には、本研究で課題としている、コンピュータ資源間のロードバランスの乱れによる性能低下問題、データステージング回数の問題によるジョブの長時間化やネットワークトラフィックの増加問題を扱った研究はなかった。

第 3 章　　インターオペレーションアーキテクチャの提案

本章では、本研究の第一の課題である異種グリッド間インターオペレーションを可能とするミドルウェアのアーキテクチャを提案する。インターオペレーションアーキテクチャの検討には、まず既存の単一グリッド環境とインターオペレーション環境でのジョブ投入・実行動作の違いを明らかにする必要がある。そこで、単一グリッドにおける単一ジョブの投入から実行終了までの動作ステップにおいて必要な機能や情報を詳細に検討し、各動作ステップにおいてジョブ投入先が他のグリッド環境の場合や、他のグリッド環境からジョブ投入を受けた場合、つまりインターオペレーション状況で、相手のグリッド環境との間でこの動作ステップが必要な機能を実現できるか、相手グリッドからこの動作ステップが必要としている情報を得られるかを検討することによって要件を明らかにし、インターオペレーションアーキテクチャの提案を行う。また、この要件を基に HPCBP 仕様のインターオペレーションの問題を明らかにし、解決案の提案、実現方法の検討、ミドルウェア実装の説明を行う。さらに、実装したミドルウェアを使用してインターオペレーション環境を構築し、他グリッドとの実証実験を行ったので、その結果を報告する。

3.1.　インターオペレーション環境におけるジョブ投入・実行

本節では、一般的な構成の単一グリッド環境おけるジョブの流れと、その各動作ステップにおける必要な情報、機能を説明する。そして、複数の単一グリッド間でジョブ実行サービスレベルインターオペレーションを実現するための要件を明確にする。

● **単一グリッドにおけるジョブ投入・実行**

一般的な構成の単一グリッドミドルウェアにおける、ジョブの投入・実行の流れを説明する。グリッドミドルウェアの構成は図 1-1 に示すような 3 階層モデルとし、実運用レベルのグリッドとして最小限のモジュールと考えられる次のモジュールから構成されるものとする。ユーザがジョブ投入操作をするユーザクライアント、ジョブを投入するコンピュータ資源を決定するメタスケジューラ、ジョブの実行要件と合致する資源を探す資源ブローカ、コンピュータ資源の情報を収集、管理する情報サービスとそのエージェント、そしてコンピュータ資源上のジョブ実行サービス、バッチスケジューラ、計算ノードである。なお、メタスケジューラやジョブ実行サービスの認証クレデンシャルは X.509 プロキシ証明書を使用するものとする。構成を図 3-1 に示す。NAREGI や gLite などは、おおよそこのような構成になっている。

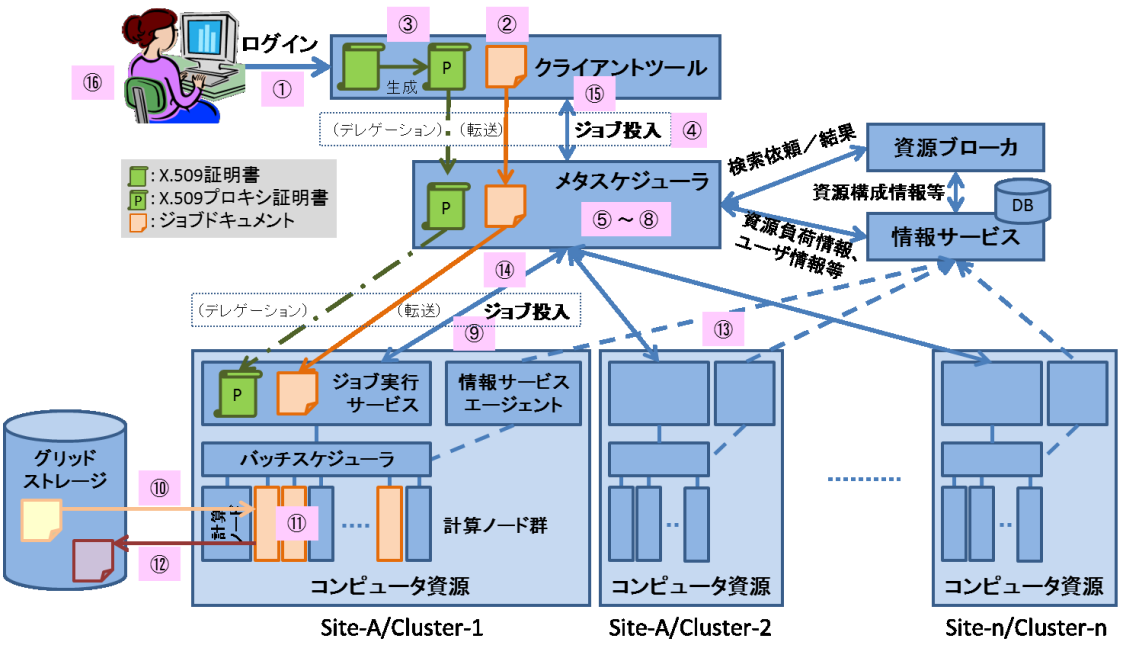


図 3-1　一般的な構成の単一グリッドでのジョブ投入動作

ユーザはジョブ投入時にコンピュータ資源を指定することもできるが、資源ブローカによって資源を決定した方が、利便性が高く資源利用効率も高いので、資源ブローカで資源を決定するものとする。なお、情報サービスはユーザが使用を開始する前からグリッド内の各種最新情報をバックグラントで収集しているものとする（図中の長い点線）。また、クライアントツールにはユーザの X.509 証明書があらかじめ配置されているものとする。この X.509 証明書は有効期限毎に更新するが、通常は年単位など長期であるため、以下のス

テップは更新時期にあたらないものとする。単一グリッド環境でのジョブ投入・実行の流れは次のようになる。各ステップ番号は図中の番号に一致する。

① ユーザはポータルなどのクライアントにログインする
② ユーザはジョブとして実行するアプリケーションのロケーション（パス）、実行に必要な資源の要件、入力ファイルのロケーション、結果の出力ロケーションなどを記述したジョブドキュメントを用意する。これはクライアントツール上で作成するか、外部から転送する。なお、資源ブローカリングの使用を前提とする場合、ジョブドキュメントには特定のコンピュータ資源の情報は記述しない
③ メタスケジューラへのジョブ投入準備として、X.509 証明書から X.509 プロキシ証明書を生成する。このとき X.509 証明書に対応するパスフレーズが必要となるが、ユーザがログインしているツールなので、ユーザに入力を促すことができる（インタラクティブ操作）
④ メタスケジューラにジョブを投入する。これは X.509 プロキシ証明書のデレゲーションとジョブ投入オペレーション（ジョブドキュメントの転送を含む）から成る
⑤ メタスケジューラはデレゲートされた X.509 プロキシ証明書を確認し、本グリッドの使用可否を判断する。使用可であれば、ジョブドキュメントを解釈し資源ブローカに資源の検索を依頼する
⑥ 資源ブローカはジョブ要件に合う資源を探し 0 個以上の資源候補をメタスケジューラに返す。このとき検索に必要な各コンピュータ資源の資源構成情報は情報サービスより取得する
⑦ メタスケジューラは資源ブローカから返ってきた資源候補から、ロードバランスなどを考慮して最適な資源を選択する。このとき各コンピュータ資源の負荷情報は情報サービスから取得する。
⑧ メタスケジューラはジョブドキュメント内の抽象化されている情報を、選択したコンピュータ資源に適合した具現化を行う
⑨ コンピュータ資源へジョブを投入する。ジョブ投入は④と同様に X.509 プロキシ証明書のデレゲーションとジョブ投入オペレーション（ジョブドキュメントの転送を含む）から成る。デレゲーション機能を用いることにより、この段階でのユーザのパスフレーズ入力は不要とすることができる
⑩ コンピュータ資源は投入されたジョブの実行を開始する。まずデータステージインが指示されている場合は、指定のロケーションから指定のファイルを指定のプロトコルで転送する。グリッドミドルウェアによっては、このジョブで実行するアプリケーション自体を転送することも可能である
⑪ 次にコンピュータ資源は、コンピュータ資源の指定されたロケーションに置かれているアプリケーションを実行する。このアプリケーションはあらかじめインストー

ルされているかもしれないし、⑩でジョブの一部として転送されたものかもしれないし、アプリケーションデプロイメントサービスなどで動的に転送されたものかもしれない

⑫　アプリケーションの実行が終了すると、コンピュータ資源は計算結果や stdout、stderr をデータステージアウト先として指定されたロケーションに、指定されたファイル名で、指定されたプロトコルで転送する（データステージアウト）

⑬　コンピュータ資源は資源の使用状況（使用 CPU 時間、使用資源量など）を記録する。この記録は情報サービスエージェント経由で情報サービスに報告され蓄積される

⑭　コンピュータ資源はジョブの終了をメタスケジューラに通知する

⑮　メタスケジューラはジョブの終了をクライアントツールに通知する。

⑯　コンピュータ資源の提供者は情報サービスに記録されているユーザの資源使用状況に従い、ユーザに使用料を請求する

● ジョブインターオペレーション

次に、インターオペレーションを実現するのに必要な機能や情報を、上記単一グリッドのジョブ投入シーケンスに沿って検討する。図 1-2 からわかるように、インターオペレーションでジョブを投入するポイントとして、メタスケジューラとジョブ実行サービスの 2 種類が考えられる。しかしメタスケジューラは、各グリッドが特長とする固有機能を実現しているグリッド資源アクセス層の入口であるため、クライアントとの間で独自機能固有の情報交換やオペレーションを持つ場合がある。このため、一般的にあまり高機能とはならないジョブ実行サービスをインターオペレーションにおけるジョブ投入ポイントとする。以下に検討結果を、上記単一グリッドのジョブ投入・実行ステップ毎に対応させて説明する。ステップ番号は単一グリッドのステップ番号に一致している。

①　ポータルなどクライアントへのログインは単一グリッドの場合と同じである

②　ジョブドキュメントの作成において、ブローカリング機能によってローカルグリッド資源、他グリッド資源のいずれが選択されてもよいように、資源に依存する情報は抽象化されている必要がある。単一グリッド環境では、すべてのコンピュータ資源で共通化のされているため抽象化されていない項目や、暗黙の合意として定義されていない項目が存在するが、これらの項目は他グリッドのコンピュータ資源では値が異なる場合があるためである。例えばデータステージングの転送プロトコルやクレデンシャルの認証方法などである

③　パスフレーズの入力は単一グリッドの場合と同じである

④　メタスケジューラへのジョブ投入は単一グリッドの場合と同じである

⑤　メタスケジューラにおける認証作業とブローカへの依頼などは、単一グリッドの場合と同じである

⑥　ローカルグリッドの全コンピューティング資源の構成情報は、まとめて情報サービスが保持している。装置の故障や資源の部分増設、部分撤去などを考慮すると、コンピューティング資源の構成情報はいつ変更になるかもしれない半静的情報である。資源ブローカに提供する構成情報はその時点の最新の情報である必要性から、情報サービスはエージェントを介して常に構成情報を収集している。他グリッドのコンピュータ資源もローカルグリッドのコンピュータ資源と同様にブローカリング可能とするには、情報サービスはその構成情報も常に収集し資源ブローカに提供する必要がある（インターオペレーションに参加しているコンピュータ資源のみ)。また単一グリッドでは、アーキテクチャ上統一されて暗黙的に使用されている項目が他グリッドでは異なる場合があるため、情報サービスはこのような資源のプロパティ情報も収集、提供する必要がある。資源ブローカはジョブ要件だけでなく、このような資源のプロパティ情報も検索条件とする必要がある。例えば、データステージングプロトコル、認証クレデンシャルタイプなどである

⑦　メタスケジューラによるジョブ投入資源の決定のために、情報サービスは⑥と同様に常に他グリッドのコンピュータ資源の負荷情報を収集、提供する必要がある（インターオペレーションに参加しているコンピュータ資源に関する情報のみ)

⑧　メタスケジューラは、ジョブドキュメントのインターオペレーション固有の抽象化された項目も具現化する必要がある。例えば、データステージングプロトコルである。さらに、ローカルグリッドのジョブドキュメント仕様と他グリッドのドキュメント仕様が異なる場合、変換が必要となる

⑨　ジョブ投入にはユーザ認証が要求される。インターオペレーション仕様で認められている認証クレデンシャルタイプがローカルグリッドと同じであれば問題ないが、異なる場合はメタスケジューラがインターオペレーション仕様で認められている認証クレデンシャルをサポートする必要がある。また、インターオペレーション仕様として複数の認証クレデンシャルタイプを認めるなら、どの認証クレデンシャルをサポートしているかを、⑧の資源プロパティ情報に含める必要がある。さらに、グリッド環境のシングルサインオンをセキュアに実現するには、認証クレデンシャルのデレゲーション機能が重要になってくる

⑩　データステージインは一般的に単一グリッドの場合と同じであるが、指定されたプロトコルを使用する必要がある

⑪　コンピュータ資源にあらかじめインストールしてあるアプリケーション以外を使用する場合、他グリッドのコンピュータ資源に如何にアプリケーションをデプロイするかが問題となる

⑫　データステージアウトは一般的に単一グリッドの場合と同じであるが、指定されたプロトコルを使用する必要がある
⑬　使用時間などの資源利用情報はコンピュータ資源の環境の情報サービスに一旦蓄積されるため、単一グリッドの場合と同じである
⑭　メタスケジューラに対するジョブの終了報告は単一グリッドの場合と同じである
⑮　クライアントに対するジョブの終了報告は単一グリッドの場合と同じである
⑯　使用料は運用情報でありジョブ投入・実行機能に直接影響を与える機能ではない。このため、特にインターオペレーション機能として実現する必要はないが、実現できればローカルグリッドと同様の費用状況確認ができ、またこれによりジョブ投入の可否を制御できるので利便性は向上する

これら必要項目を実現するために、仕様化が必要な項目を抽出する。ステップ①～⑤、⑩、⑫～⑮は各グリッド閉じた項目であり、インターオペレーションに関連する項目であっても各グリッド固有の仕様で問題ない。⑥では資源構成情報、資源プロパティ情報の仕様化が必要である。また例として挙げたデータステージングプロトコルや認証クレデンシャルタイプは、すべてのグリッドがサポートしている、すべてのプロトコルやタイプをサポートすることは難しいため、種類を絞り込むための仕様化が必要である。⑦では資源負荷情報のグリッド間での交換が必要であり仕様化が必要である。⑧ではインターオペレーションプロファイル仕様の JSDL エレメントの準拠規定の厳密化や、資源プロパティなどを定義する JSDL のインターオペレーション拡張仕様の策定が必要となる。⑨では認証クレデンシャルのデレゲーションとジョブ投入オペレーションの仕様化が必要である。⑪ではグリッド間のアプリケーションデプロイメントの仕様化が必要である。⑯では資源使用情報の仕様化が必要となる。

まとめると、単一グリッドからインターオペレーションに移行するためには、少なくとも以下の項目の仕様化が必要であることが判明した。

- データステージングプロトコル仕様
- ジョブ記述仕様
- 認証クレデンシャルのセキュアなデレゲーション仕様
- ジョブ投入オペレーション仕様
- 資源構成情報、資源プロパティ情報、資源負荷情報、資源使用情報の仕様ならびにこれらの情報の交換仕様
- アプリケーションデプロイメント仕様

3.2.　インターオペレーション仕様とアーキテクチャの提案

なるべく多くのグリッドミドルウェアがインターオペレーション対応するには、仕様が

国際標準であること、インターオペレーションアーキテクチャが既存の単一グリッドミドルウェアアーキテクチャからかけ離れていないこと、できれば既存のアーキテクチャの延長線上で容易に拡張できるアーキテクチャであること、また対応工数が少ないことが重要であると考える。そこで、単一グリッドミドルウェアとして一般的な図 3-1 のアーキテクチャを持ち、上記インターオペレーションとして必要な項目を実現するアーキテクチャを検討した。まず、上記必須項目に対し既存の標準仕様を適用した場合の問題点とその提案解決方法の概略を説明し、次にアーキテクチャの提案を行う。

3.2.1. インターオペレーション仕様

- **ジョブ記述仕様**

ジョブ記述の標準仕様としてよく採用されているのが、OGF の JSDL である。しかし、JSDL はジョブ記述テンプレートとしてポータビリティを重視した設計思想のため、ジョブ実行時や環境により異なる項目に対する記述が弱い。例えばバッチスケジューラのキュー名は指定できない。また、JSDL は単一アクティビティ（本議論ではジョブと同意と考えて良い）の記述しかできずワークフロージョブの記述はできない。このため、JSDL を中核としたワークフロージョブの記述言語が必要と考える。例えば、Business Process Execution Language for Web Service（BPEL） [43]などが候補として挙げられる。また、科学計算を考慮すると、並列ジョブやパラメータサーベイ型のジョブの記述が必須である。このためジョブ記述として、JSDL HPC Profile Application Extension、JSDL SPMD Application Extension [44]、JSDL Parameter Sweep Job Extension、または類似の拡張仕様のサポートは必須と考える。

- **ジョブ投入オペレーション仕様**

ジョブ投入の標準仕様については、HPCBP や PGI によるインターオペレーションでは OGF の BES が採用されている。BES はジョブ投入オペレーションの基本的な機能しか持っていないが、各グリッドミドルウェアが持つ豊富な機能をインターオペレーション環境で実現するには、各グリッドの対応工数が大きくなり、また整合性の問題などが多く発生することが予想されるので、基本機能で十分であると考える。ただし、単に BES の機能だけでは、第 4.2 節で述べるデータステージング回数の問題などが発生するため、PGI 要件への対応を検討している版数の BES 仕様に準拠する必要がある。

- **認証クレデンシャルのセキュアなデレゲーション仕様**

コンピュータ資源へのシングルサインオンを実現するには、認証クレデンシャルのセキュアなデレゲーション機能や他の委譲機能が必要である。多くのグリッドが Grid Security Infrastructure（GSI） [45] 仕様を採用している。単一グリッド環境では、一人のユーザの証明書は一種類であることを前提とできるが、インターオペレーション環境では複数の X.509 証明書が必要になる場合も考えられる。これは、デレゲーション機能に限らず、元の X.509 証明書自体を使用する場合にも問題となる。これを解決するには、各グリッドがどこの CA 局が発行した X.509 証明書が利用できるかを、あらかじめ情報交換する仕様が必要である。

- **データステージングプロトコル仕様**

データステージングのプロトコル仕様としての選択肢は多く、セキュアな方式を選択する。GridFTP、ByteIO、SSH（SFTP, SCP）、HTTPS などがあり、参照実装も存在するため、各グリッドミドルウェアへの実装は難しくないと考えられる。しかし、これらのプロトコルは複数の認証方式を認めているため、データステージングを行う資源間で同じプロトコルをサポートしていても、認証方式の違いからエラーになることもある。そこで、サポートする認証方式をあらかじめ情報交換する必要がある。

- **アプリケーションデプロイメント仕様**

商用 OS や Linux のディストリビューションでは、アプリケーションをインストールするためのデプロイメント仕様が存在し広く利用されているが、異グリッド間でのデプロイメントを意識した仕様はないと考えられる。そこで、アプリケーション一式をアーカイバでまとめてデータステージングで転送し、JSDL のデータステージングエレメントに、展開やファイル属性の変更を指示する抽象表現のエレメントを仕様化することを提案する。

- **情報の交換仕様**

情報の種類によって複数の情報表現仕様を採用する必要がある。従来のグリッドミドルウェアにおいて、コンピュータ資源の構成や負荷情報などについては、OGF の GLUE2 [46] に、資源使用情報については OGF の UsageRecord（UR） [47]に準拠することが多い。上記のように、単一グリッド内では統一されていた機能や仕様が、インターオペレーション環境では様々であるため、採用している機能や仕様をグリッド間で情報交換する必要がある。GLUE2 には、定義されていない情報もあるため拡張が必要である。

3.2.2.　インターオペレーションアーキテクチャの提案

上記仕様のうち、ジョブ投入オペレーション仕様と情報の交換仕様以外は、各グリッド内の既存のサービスやモジュール上で対応すべき仕様であり、インターオペレーションアーキテクチャとしての新たな接続やモジュールの追加はない。ジョブ投入オペレーションについては、当然ながら他のグリッドへのジョブ投入をするための新たな接続が必要となる。これは図 1-2 のインターオペレーションの概念アーキテクチャで示したグリッド間の接続である。情報交換でもグリッド間で新たな接続が必要となる。単一グリッド内での情報交換は情報サービスが集中的に行うが、インターオペレーションではグリッドをまたぐような中央情報サービスのようなものは望みにくく、各グリッドの情報サービス同士が通信して情報を交換するのが自然と考える。図 3-2 に提案のインターオペレーションアーキテクチャを示す。なお、図には情報サービスと記述しているが、他のサービスが情報サービスと同等の機能を持つ場合もあり、またジョブのインターオペレーションのポイントがコンピュータ資源になっているがメタスケジューラになる場合もある。

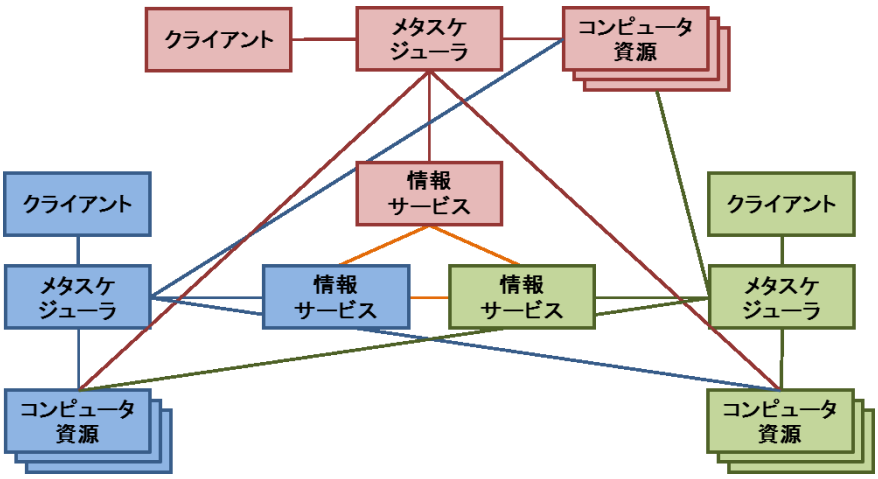


図 3-2　インターオペレーションアーキテクチャ

3.3.　HPCBP 仕様によるインターオペレーション

上記インターオペレーションアーキテクチャは、すべてのグリッドがその情報サービスやジョブ実行サービスに上記標準仕様を実装することを前提としている。しかし現実には、インターオペレーションは HPCBP 仕様によるジョブ投入・実行と一部の資源情報の交換のみに留まっており、その他の上記必要項目はインターオペレーションに参加するグリッド間での合意や努力、そして機能制限により実現されている。しかし、インターオペレーションに関する研究は、このレベルでも出来る課題は多い。そこで、本研究では HPCBP

仕様によるインターオペレーションの実現も課題のひとつとした。本節では、HPCBP 仕様だけでは不足している機能や情報を明らかにし、不足していることによる問題点、その解決方法の提案を行う。本課題の HPCBP 仕様によるインターオペレーションは、著者が国立情報学研究所において研究開発した NAREGI グリッドミドルウェアをベースに実現する。このため、NAREGI 仕様と HPCBP 仕様との間の整合性の問題が発生し、この解決方法の提案も行う。なお、HPCBP 準拠のインターオペレーションミドルウェアの多くは、ベースとなるミドルウェアとの間で仕様の整合性の問題を持っており、各自の努力で解決し、他グリッドとのインターオペレーションに極力支障がでないようにしている。

3.3.1. HPCBP で不足している仕様

第 3.1 節で説明したインターオペレーションに必要な仕様のうち、HPCBP では、ジョブ記述仕様として JSDL 仕様を、ジョブ投入オペレーション仕様として BES 仕様を、認証クレデンシャル仕様として X.509 証明書仕様またはユーザ名／パスワードトークン仕様を定義しているだけである。またグリッド間の情報交換として、BES 仕様で資源構成情報、資源負荷情報の一部を交換可能であるが、その他の情報の交換仕様については定義されていない。これら問題のため、実際には HPCBP を実装しただけでは、他のグリッドと実験レベルのインターオペレーションさえも実施することはできない。これらの仕様が不足していることによる問題点について説明する。

- **グリッド間情報交換仕様**

第 3.1 節で説明したように、インターオペレーションを実現するには、資源構成情報、資源負荷情報、資源プロパティ情報、資源使用情報の仕様、ならびにこれら情報の交換仕様が必要である。これら情報が不足していることによる問題点を説明する。まず資源ブローカが参照を必要としている資源構成情報について検討する。BES の *GetFactoryAttributesDocument* オペレーションを使用すると、コンピュータ資源から表 3-1 に示す資源情報を取得することができる。この中には資源の構成情報として、*LocalResourceManagerType*、*ResourceName*、*OperatingSystem*、*CPUArchitecture*、*CPUCount*、*CPUSpeed*、*PhysicalMemory*、*VirtualMemory* があるが、JSDL として指定できるリソース条件を網羅していないため、資源ブローカリング結果が有効でない場合がある。例えば JSDL の *NetworkBandwidth* や *DiskSpace* 関連のアトリビュートがないため、通信性能を重視するアプリケーションや巨大なデータを扱うアプリケーションに対して有効なブローカリングができないという問題がある。特に巨大データを扱うアプリケーションの場合、十分なディスクスペースがある資源で実行しないとエラーが発生する可能性が高い。なお、並列ジョブのための資源をブローカリングするのに必要なノード数やノー

ドあたりの CPU 数などが見当たらないが、これは FactoryAttribute が階層構造になっているためである。表 3-1 の情報がノード単位とクラスタ単位で階層的に表現されており、これを解析すればノード数やノードあたりの CPU 数などを得ることができる。このため、現状の仕様でも並列ジョブの資源ブローカリングは可能である。

次にメタスケジューラが参照する資源負荷情報について検討する。資源ブローカから戻ってくるジョブ投入候補資源が複数の場合、メタスケジューラはその中から 1 つを選択するが、その選択基準のひとつが候補資源の負荷状況である。*GetFactoryAttributesDocument* オペレーションで得られる負荷情報には、処理中のアクティビティの総数を表す *TotalNumberOfActivities* のみしかなく、実行中のアクティビティ数と実行待ちのアクティビティ数が不明であるため、適した資源の選択ができない。やむなく無作為に資源を選択すると、実行待ち時間の長期化、資源の利用効率の低下などの問題が発生する。*ActivityReference* でアクティビティの End Point Reference（EPR）リストを取り出し *GetActivityStatuses* オペレーションで各アクティビティのジョブステートから実行状態であるか実行待ち状態であるかを判定することも考えられるが、特権ユーザの仕様がないため、自分で投入したジョブ以外では認証エラーが発生する。このため、この方法は使えない。

次に資源プロパティ情報について検討する。資源には様々なプロパティがあるが、上記ジョブ投入・実行の流れで問題となるのは、データステージングプロトコルとして何が使用できるか、認証クレデンシャルは何を使用しているか、認証方法は何かの情報が必要である。これらの情報は単一グリッド内では統一されている場合がほとんどなので資源ブローカリングで任意の資源が選択されても問題なかったが、他のグリッドとは異なる可能性があり、異なるプロパティを持つ資源にジョブを投入するとエラーとなる。このため、これら情報をブローカリングの検索条件とする必要があり、あらかじめサポート情報を資源プロパティ情報として交換しておく必要がある。

次に資源の使用情報であるが、これは運用として必要な情報であり、一般にはジョブ投入としてのインターオペレーションの一部としては交換する必要はない。このため、本論文ではこれ以上の議論は行わない。

具体的な解決策として、様々な情報を交換できる仕様がない以上、BES の資源情報だけはインターオペレーション環境で交換することとし、他の必要な情報はインターオペレーション仕様外で交換するしかない。後述のグリッド間で交換が必要な情報はすべて静的な情報であるため、あらかじめファイル化して交換しておき、これを必要なときに利用しても問題はない。ただし、上記の資源の負荷情報に関しては解決方法がなく、他グリッドのコンピュータ資源に対してはロードバランスを考慮したスケジューリングができない問題が残る。

表 3-1　BES のアトリビュート

情報名	概要
IsAcceptingNewActivities	True if BES is accepting new activities
CommonName	Short human-readable name for BES
LongDescription	Longer human-readable description
TotalNumberOfActivities	Number of activities currently managed in BES
ActivityReference	EPRs to activities currently active in BES
TotalNumberOfContainedResources	Number of contained resources accessible by the BES
ContainedResource	Currently expected to be either of type BasicResourceAttributes DocumentType or FactoryResourceAttributesDocumentType
NamingProfile	URIs of Naming profiles used by BES
BESExtension	URIs of supported BES extensions
LocalResourceManagerType	Resource's local resource manager type
ResourceName	Resource's name
OperatingSystem	Resource's operating system type
CPUArchitecture	Resource's CPU architecture type
CPUCount	Resource's CPU count
CPUSpeed	Resource's CPU speed in Hertz
PhysicalMemory	Resource's physical memory in bytes
VirtualMemory	Resource's virtual memory in bytes

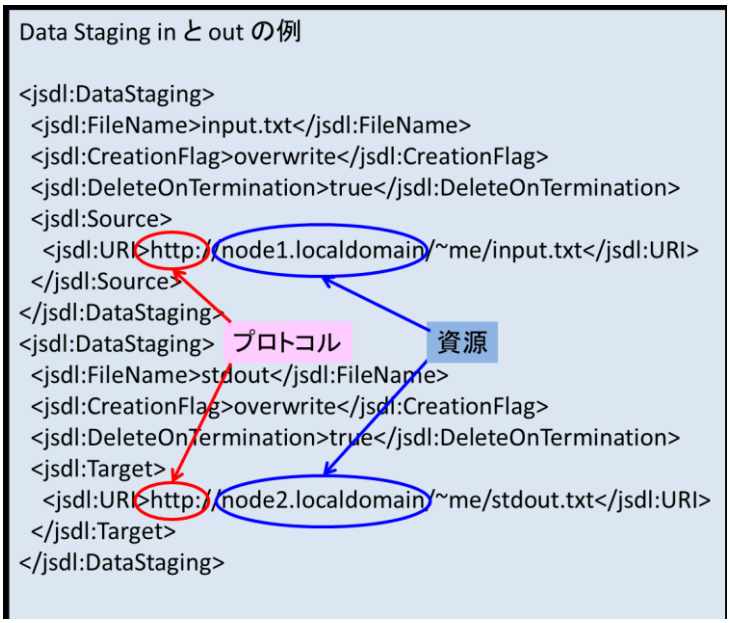
- **データステージングプロトコル仕様の欠如**

データステージングとは、ジョブ実行に必要な入力データをコンピュータ資源へ転送し、また実行結果（計算結果、stdout、stderr）をコンピュータ資源から外部へ転送することを指す。このデータステージングに使用するプロトコルはグリッドにより異なり、NAREGI や gLite のコンピュータ資源では GridFTP が、XSEDE のコンピュータ資源では ByteIO が主に採用されている。従来の単一グリッド環境ではすべてのコンピュータ資源が同じプロトコルをサポートしており、データステージングプロトコルの違いによるエラーは発生しなかった。しかし、インターオペレーション環境ではグリッド間でプロトコルが異なることがあるため、ジョブは投入できてもデータステージングでプロトコルの違いによりエラーとなり実行できない問題が発生する。

さらに、プロトコルの違いは認証の問題も引き起こす。基本的にデータステージングには認証が必須であり、同じプロトコルでも複数の認証方式をサポートする場合がある。例

えば、グリッド環境で多く利用されている GridFTP は X.509 プロキシ証明書による認証を行うのが一般的であるが、ユーザ名／パスワード認証も可能である。また、同じ認証方式を使用する場合も使い方によって交換する情報が異なる場合がある。たとえば GridFTP ではデレゲーションプロトコルにより X.509 プロキシ証明書をデレゲートする場合と、X.509 プロキシ証明書を格納している MyProxy サーバの URL とユーザ名／パスフレーズの情報交換を行い、GridFTP を起動する前に MyProxy サーバから X.509 プロキシ証明書を取得してから実行する例もある。このように、この問題は単にデータステージングプロトコルのサポート情報を交換するだけでは解決しない。

この問題を解決するには、以下の対策が必要と考える。①上記「グリッド情報交換仕様」の項で説明したように、サポートするデータステージングプロトコル、認証クレデンシャルタイプの情報、認証方式の情報を交換し、資源ブローカリング際の検索条件の一部とする。これには、情報交換仕様がないことが問題となるが、これらの情報は静的な情報であり、インターオペレーション以外であらかじめ情報交換をしておき、これを利用することで解決できる。②ユーザが作成するジョブドキュメントにおいて、データステージングプロトコルなどグリッドにより異なる仕様を採用している可能性がある項目は、抽象表現することを可能とし、メタスケジューラなどがこれを具現化する。例えば、ジョブドキュメントが JSDL で記述される場合、データステージングプロトコルは *DataStaging/URI* エレメントに URI の記法に則り、プロトコルとデータソースまたはデータターゲット資源を記述する（図 3-3 参照）。これを抽象表現可能な記述仕様とする。抽象表現としては、「一般的に出現しないようなエスケープ開始文字列」＋「抽象化するエレメント種別」＋（「シーケンス番号」＋）「一般的に出現しないようなエスケープ終了文字列」とする。例えばユーザはプロトコル部を”@@REPLACE-DataTransferProtocol-REPLACE@@”、コンピュータ資源部を”@@REPLACE-ComputingResource-REPLACE@@”などのように記述する。これらの抽象化された各情報は、メタスケジューラがジョブを投入するコンピュータ資源を決定したときに具現化する。



（出典： “Job Submission Description Language (JSDL) Specification”, [24]）

図 3-3　*DataStaging* エレメントの記述例

- **認証クレデンシャルのセキュアな委譲仕様の欠如**

HPCBP では認証クレデンシャルとして、X.509 証明書とユーザ名／パスワードトークンを仕様化している。しかし、主要なグリッド環境では図 3-1 に示すようにユーザが操作するクライアントとコンピュータ資源やストレージ資源の間にはグリッド資源アクセスサービス群が存在し、これらサービスが資源をアクセスする際にも認証が必要となる。このため、X.509 証明書を認証に使用する場合には、証明書とパスフレーズをサービス群に転送する必要がでてくる。例え、暗号化された通信チャネルを使用して転送するとしても、今日ではこのような認証クレデンシャルの転送は安全ではないと考えられている。

HPCBP で他のグリッドとのインターオペレーションのためには、X.509 証明書と対応するパスフレーズを、ジョブを投入するクライアント（例えば、メタスケジューラ）に持つしかない。解決方法は、各グリッドミドルウェアの実装に依存し一般解はないため、RENKEI での解決方法を後述する。

- **ユーザアプリケーションのデプロイメント仕様の欠如**

グリッド上のコンピュータ資源でジョブとしてユーザアプリケーションを実行するには、ジョブの投入時または事前にユーザアプリケーションをコンピュータ資源に転送し、実行属性を付加しておく必要がある。各グリッドミドルウェアは、独自のユーザアプリケー

ションのデプロイメント機能または方針を持つが、HPCBP では規定されていない。データステージング機能を利用して、ジョブ内でアプリケーションをコンピュータ資源に転送することはできるが、実行属性を付加する方法がなく、JSDL の *Executable* エレメントにこのアプリケーションを指定しても実行できない問題がある。

この問題を解決するには、現状ではアプリケーションをあらかじめコンピュータ資源にインストールしておくしかない。しかし、グリッドによっては、ジョブ毎に動的にユーザアカウントと作業領域を割当てる一時アカウント管理を行っている場合があり、ジョブ実行時以外には個々のユーザのアカウントが存在せず、また home のような恒久的な作業領域もないので、あらかじめインストールするにはコンピュータ資源管理者への依頼が必要になるという問題が残る。

3.3.2. ベースミドルウェア仕様との整合性

既存の独自仕様ミドルウェアに HPCBP 準拠のジョブ実行サービスを追加して、インターオペレーション対応のミドルウェアとする場合、両者の基本機能仕様が異なり整合性の問題が発生することがある。例えば、認証クレデンシャル仕様、データステージング仕様などの違いが挙げられる。これら個々の基本機能仕様の違いはベースとするグリッドミドルウェアによって異なるが、最も共通に問題になるのが認証クレデンシャルの違いである。多くのグリッドミドルウェアは証明書のデレゲーション機能を採用しており、コンピュータ資源での認証確認にはデレゲートされた認証クレデンシャル（プロキシ証明書）を使用する。各ミドルウェアは各々のセキュリティポリシーを基に基本アーキテクチャを設計しているため、HPCBP 仕様に合せるためにミドルウェア全体の認証クレデンシャルを、デレゲーション機能のない x.509 証明書に変更することは、ポリシーや設計の根幹に影響する大きな問題となるため一般には認められない。

基本的な解決策はデレゲーション仕様を定義することであるが、これには標準化議論に時間を要する。短期的な解決策としては、HPCBP ジョブ実行サービスにジョブを投入するときのみ X.509 証明書を使用する方法が考えられる。ジョブ投入直前までは、従来の認証クレデンシャルを使用し、ジョブ投入時に X.509 証明書を基にした認証に変更するのである。多くの場合メタスケジューラにこの変更の仕組みを持つことになる。しかしこのとき、メタスケジューラに上記「認証クレデンシャルのセキュアな委譲仕様の欠如」で説明したように、X.509 証明書とパスフレーズを転送すると、ミドルウェアのセキュリティポリシーに反する可能性がある（例えば、認証クレデンシャルのネットワークでの転送禁止）。そこで、解決方法としてメタスケジューラ上に各ユーザの X.509 証明書とパスフレーズを格納するセキュアなリポジトリを設け、デレゲートされた認証クレデンシャルを鍵にこれらを取り出すといった手法が考えられる。ただし、認証局のポリシーによっては X.509 証明書

のコピーを禁じている場合もあるので、この場合はこの解決策は使えない。

これら基本機能の準拠仕様の違いの問題は数多くあると考えられるが、問題点と解決方法はグリッドミドルウェアにより異なり一般解はない。このため、本節ではこれ以上の説明や解決案の検討は行わない。次節以降に NAREGI ミドルウェアに HPCBP 仕様準拠のジョブ実行サービスを追加する説明を行うが、このときに NAREGI ミドルウェアの場合の問題点と解決方法について議論する。

3.4. HPCBP インターオペレーションミドルウェアの研究開発

本節では、本研究の課題の一部であるインターオペレーションミドルウェアの研究開発について説明する。第 3.2 節で説明したように、インターオペレーション対応ミドルウェアは従来のグリッドミドルウェアに HPCBP 仕様準拠のジョブ実行サービスを追加したものが多い。本研究でも国立情報学研究所が開発した NAREGI ミドルウェアに HPCBP 準拠サービスを追加することにより実現した。本節では、その基本要件、問題点と解決策、設計概要、他グリッドとの実証実験について説明する。

なお、説明上の混乱と記述の煩雑さを避けるために、本論文では HPCBP 仕様に準拠した NAREGI ミドルウェアを RENKEI ミドルウェアと呼び、NAREGI と HPCBP の両仕様のジョブ投入を受け付け可能なコンピュータ資源を RENKEI コンピュータ資源、NAREGI 仕様のジョブのみ受け付け可能なコンピュータ資源を NAREGI コンピュータ資源、HPCBP 仕様のジョブのみ受け付け可能コンピュータ資源を HPCBP コンピュータ資源と表記する。また、他グリッドの HPCBP 仕様のジョブを受け付け可能なコンピュータ資源は、その他の仕様のジョブを受け付け可能であっても単に HPCBP コンピュータ資源と表記する。さらに、NAREGI ミドルウェアのサービスやモジュール群は NAREGI-｛サービス・モジュール名｝と表記し、RENKEI ミドルウェアのサービスやモジュール群は同様に RENKEI-｛サービス・モジュール名｝と表記する。

3.4.1. RENKEI プロジェクトのインターオペレーション要件

本研究のインターオペレーションミドルウェアは、RENKEI プロジェクトの研究の一環として開発した。このため、単にインターオペレーション動作が可能なミドルウェアを実現するだけではなく、プロジェクトが設定したいくつかの要件を満たす必要があった。以下にその要件を列挙する。

① NAREGI コンピュータ資源、RENKEI コンピュータ資源、HPCBP コンピュータ資源を一つのワークフロージョブ内で混在使用可能なこと〔ジョブ投入要件〕

② NAREGI コンピュータ資源、RENKEI コンピュータ資源、HPCBP コンピュータ資源で極力同一のジョブ記述が使用できること〔ジョブ投入要件〕〔ジョブ実行要件〕
③ NAREGI コンピュータ資源と同様に RENKEI コンピュータ資源と HPCBP コンピュータ資源も資源ブローカリング可能なこと〔ジョブ投入要件〕
④ RENKEI コンピュータ資源、HPCBP コンピュータ資源に対してもシングルサインオンが可能なこと〔ジョブ投入要件〕
⑤ RENKEI 環境と HPCBP コンピュータ資源間でデータステージング可能なこと〔ジョブ投入要件〕〔ジョブ実行要件〕
⑥ RENKEI コンピュータ資源は NAREGI 仕様によるジョブ投入インターフェースと HPCBP 仕様のジョブ投入インターフェースの両方を持つこと〔ジョブ実行要件〕
⑦ RENKEI コンピュータ資源において、NAREGI 仕様でジョブ投入した場合は NAREGI コンピュータ資源のもつすべての機能を実行できること〔ジョブ実行要件〕

3.4.2. 実装設計

NAREGI ミドルウェアのアーキテクチャを図 3-4 に示す。クライアント群はユーザ管理サービス（UMS： User Management Service）とポータルから成り、ポータルには、ジョブドキュメントの記述、ジョブ投入・管理・モニタをするためのワークフローツール（WFT : Work Flow Tool）、ユーザアプリケーションのデプロイなどを行う問題解決環境（PSE : Problem Solving Environment）などのツール群が用意されている。グリッド資源アクセスサービス群として、スーパースケジューラ（SS）と情報サービス（IS）があり、SS はメタスケジューラ、資源ブローカ、ワークフローエンジンなどからなる。情報サービスは階層構造を持っており、各コンピューティングサイトで配下のコンピュータ資源の情報を取りまとめる IS Cell Domain Aggregation Service（IS-CDAS）、各サイトの IS-CDAS の情報を取りまとめてグリッド環境全体の情報を管理する IS Node information Aggregation Service（IS-NAS）からなる。SS などのサービスやモジュールは IS-NAS から情報を取得する。IS-CDAS は IS 関連のサービスやモジュール以外からアクセスされることはないため、以下の説明では省略し、単に情報サービスや IS と表記した場合、IS-NAS を意味するものとする。コンピュータ資源には、ジョブ実行サービスである GridVM（Grid Virtual Machine）、情報サービスエージェントである IS-LRPS（IS-Local Resource information Provider Service）、バッチスケジューラ、計算ノードから構成される。

NAREGI ではすべてのジョブが Portal → SS → GridVM の流れで投入され、他のグリッドにみられるような、クライアントから直接コンピュータ資源に投入する流れはない。このため HPCBP 対応においても、ジョブ投入以外の機能との整合性を考慮すると Portal → SS → HPCBP の流れとすべきである。したがって、SS に HPCBP クライアント機能

を実装する。また、コンピュータ資源のジョブ実行サービスの HPCBP 対応については、GridVM に HPCBP インターフェースを追加する方法と、独立した HPCBP ジョブ実行サービスを併設する方法が考えられる。GridVM の互換性保証、HPCBP との認証クレデンシャルの違いを考慮すると、併設すべきと考える。図 3-5 に上記要件を満たす RENKEI ミドルウェアの基本アーキテクチャを示す。図中の赤色の太線で囲まれたモジュールが HPCBP 仕様に準拠するためのモジュールである。このうち ”*” 記号が付加されているモジュールやサービスが著者の成果であり、付加されていないものは RENKEI プロジェクトの他のメンバーによる成果である。

3.4.2.1. HPCBP 準拠クライアント、ジョブ実行サービス

HPCBP 準拠のクライアントとジョブ実行サービスは、オープンソフトである BES++ [48] [49]をベースに研究開発した。BES++の改造は、HPCBP 準拠インターオペレーションを実現するために必要な開発（BES++の障害対応も含む）と、抽出したインターオペレーションの問題点を解決するための開発がある。本章では両者を議論するが、後者は第 4 章でも議論される。以下、開発した HPCBP サービスを RENKEI-HPCBP サービスと表記する。ただし、RENKEI の説明内では単に HPCBP サービスと表記することもある。

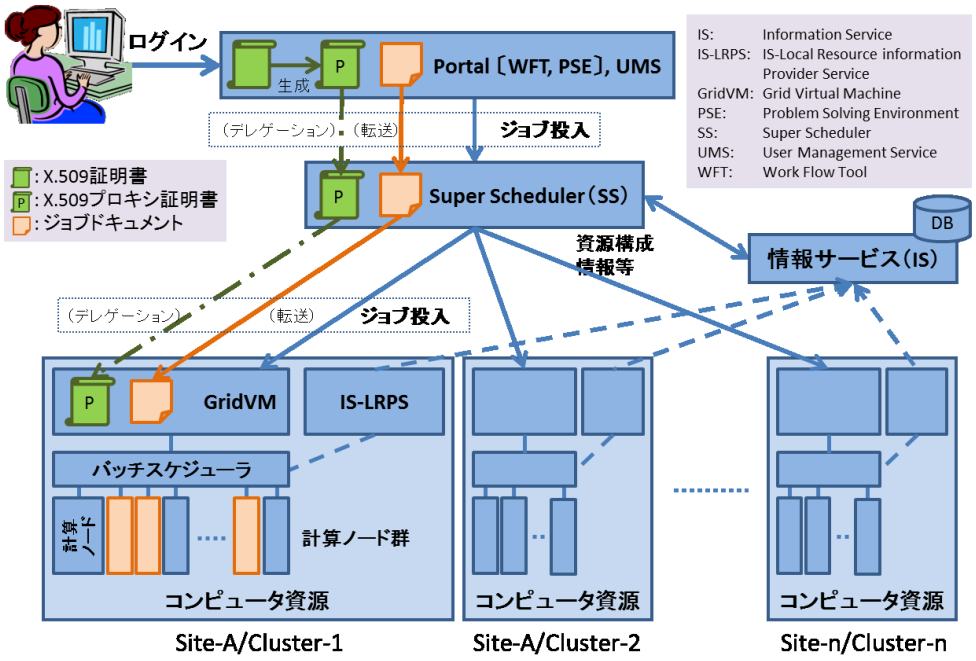


図 3-4　NAREGI ミドルウェアのアーキテクチャ

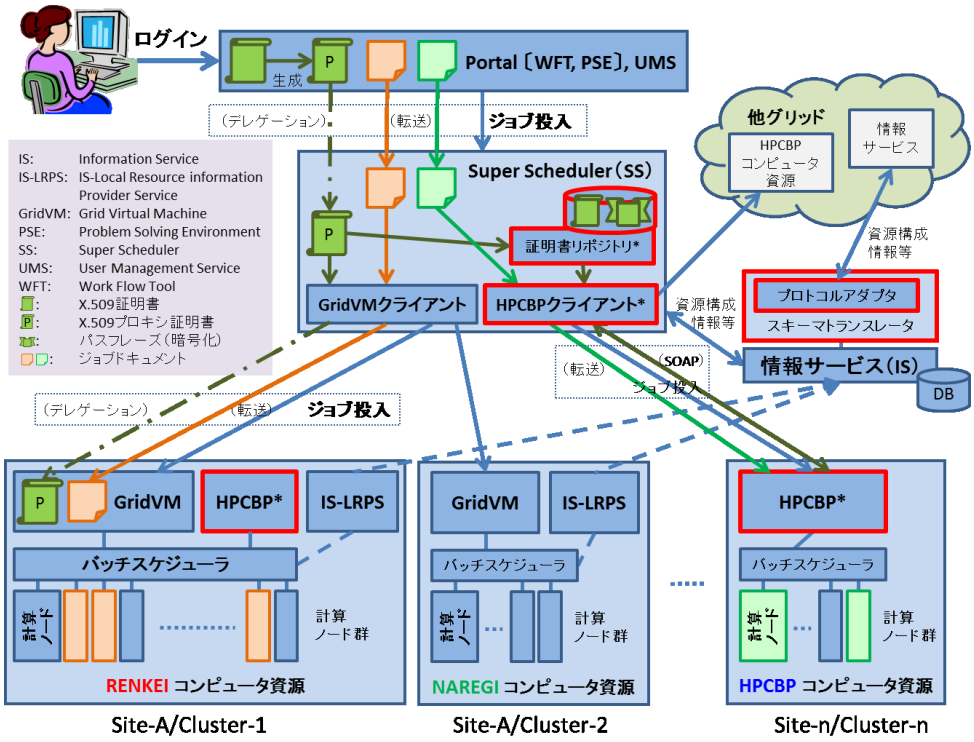


図 3-5　RENKEI ミドルウェアの基本アーキテクチャ

BES++サービスでは、実行ステータスなどジョブ実行関連情報は下位のバッチスケジューラの情報をそのまま利用している。サービス自身は情報を保持せず、情報取得要求オペレーションを受信するたびにバッチスケジューラに問い合わせている。一般に、バッチスケジューラはこれらの情報をジョブ終了後直ちに消去するため、一旦ジョブが終了すると BES++クライアントは終了コードなどの状況を取得できない。このため、ユーザはジョブが正常終了したのか、異常終了したのかの判断ができない問題があった。BES や HPCBP の仕様では、ジョブ実行終了後のジョブ実行関連情報の保持期間を規定していないためバグとは言えないが、ユーザが終了状態を確認できないのはシステムとして基本的な欠陥である。この問題を解決するため、ジョブ終了後も情報を DB 化して保持するよう改造した。しかし、情報を保持することにすると消去するトリガが必要となるが、HPCBP 仕様には消去のトリガとなるオペレーションやタイミングはない。従来のコンピュータシステムでよく使用される方法として、クライアントによって一旦情報が取得されると消去するといった方法があるが、BES 仕様に情報保持期間の定義がない以上、この方法が正しいとは言えない。クライアントの実装によっては、ジョブ情報を複数回要求する可能性もあるためである。この件は、PGI 仕様の検討においても問題視され、情報を消去するオペレーションを要件として挙げている。本研究では、実験的にジョブの実行関連情報を消去する独自仕様のオペレーションを実装した。しかし、独自仕様のオペレーションであるため、他グリッ

ドからは使用できない。このため、Linux システムの cron 機能を利用して、ジョブ終了後一定時間経過した情報は消去する仕組も組み込んだ。なお、この情報のサイズはジョブあたり 1KB 以下と小さいので、ある程度長期間消去しなくても問題はない。

その他の改造項目は、次の 4 項目である。

- ジョブ管理情報の DB 化
- RENKEI-HPCBP 再起動時における前回停止時のジョブ管理情報の復元
- GSI 互換の X.509 証明書のデレゲーション機能
- 「4.2. データステージング回数問題」の解決策の改造　他

また BES++サービスは、実行中のジョブの情報についてもメモリ上に持つだけであり、BES++サービスが正常、異常問わず終了すると情報が消えてしまう。このため、BES++サービスを停止し再起動すると、停止したときに実行していたすべてのジョブ情報がすべて不明となり、計算ノード上ではジョブが継続されているのも関わらず、クライアントからはジョブが不明になってしまう問題があった（ジョブが存在したこと自体も不明となる)。そこで、実行中のジョブの管理情報も DB 化し、再起動したときに停止時のジョブ管理情報を復元する機能を追加した。なお、復元時には終了状態以外のジョブの管理情報を最新の状況に更新する。

「3.3.1. HPCBP で不足している仕様」で説明したように、HPCBP には「認証クレデンシャルのセキュアなデレゲーション仕様」がなく、また BES++においてもデレゲーション機能の実装はない。しかし、PGI 仕様では認証クレデンシャルのセキュアなデレゲーション機能が要件として挙げられ検討されているため、RENKEI-HPCBP では先行してデレゲート機能を実装した。具体的には、GSI の X.509 証明書のデレゲーションプロトコルを実装した。実験では、認証クレデンシャルをデレゲートする仕組みとしても、問題が発生しないこと確認した。なお、他の HPCBP 準拠グリッドミドルウェアではまだ実装されていないため、インターオペレーション実験や運用ではデレゲーション機能は使用できない。

「4.2. データステージング回数問題」の解決策の改造については、第 4.2 章で説明する。

3.4.2.2. SS の HPCBP 準拠

図 3-6 に開発した RENKEI-SS の構造を示す。RENKEI-SS は上記のように NAREGI-SS を基に開発しており、図では NAREGI-SS から変更した部分に ”*” 記号を NAREGI-SS に追加した部分を”＋”記号を付加し赤色の太線で囲んで示す。RENKEI-SS として満たすべき上記要件は①～⑤である。以下、これら要件に沿って関連する問題の解決策と実装設計を説明する。なお、NAREGI-SS は OGF の Open Grid Service Architecture - Execution Management Service （OGSA-EMS）の機能を参考にアーキテクチャを設計している。

OGSA-EMS では、資源決定サービスや資源候補サービスなど、メタスケジューラとしての各種機能を個別サービスとし、ジョブマネージメントサービスがまとめてメタスケジューリングサービスとするアイデアとなっている。本節では HPCBP 準拠のために追加、改造したサービスの機能概要は説明するが、その他サービスの機能については OGSA [50] のドキュメントなどを参照されたい。

要件①「NAREGI コンピュータ資源、RENKEI コンピュータ資源、HPCBP コンピュータ資源を一つのワークフロージョブ内で混在使用可能なこと」は、RENKEI プロジェクトの研究課題であるグリッド間の資源連携を実現するための最も重要な要件である。ユーザがどの管理ドメインにあるコンピュータ資源でも自由に組み合わせて使用可能としなくてはならない。これにより、例えばユーザは手元にある実験データを NAREGI コンピュータ資源上の組織外の資源での実行が許されていないアプリケーションで処理を行い、その結果をさらに共同研究先の HPCBP コンピュータ資源のみにライセンスが限定されているアプリケーションで処理するといった資源連携が可能となる。なお、要件①はジョブ投入側（グリッド資源アクセスサービス群）の総合的な要件であり、要件②～⑤の要件が満たされていないと実現できない。そこで、まず要件②～⑤の実現方法を先に説明する。

要件②は、RENKEI-SS とコンピュータ資源の両方の対応が必要となる。コンピュータ資源側の対応については次節で説明する。RENKEI-SS は、ジョブドキュメントにおけるデータステージングの記述方法とその動作の違い、JSDL のサポートエレメントの違いに対応する必要がある。HPCBP ではデータステージングを JSDL のデータステージングエレメントに記述するのに対し、NAREGI では独立したファイルトランスファーアクティビティとして記述する。このため、RENKEI-SS はジョブドキュメントにファイルトランスファーアクティビティが含まれ、かつデータトランスファーのソースとディスティネーションのいずれかが HPCBP コンピュータ資源である場合は、ジョブドキュメントの JSDL 部にデータステージングエレメントを追記する設計とした。なお、データステージング動作の違いの対応については要件⑤で説明する。JSDL のサポートエレメントの違いについては、NAREGI ではサポートしていないが HPCBP ではサポートされている JSDL エレメントについて、SS の解析でエラーとしない設計とした。

要件③のブローカリング要件を実現するには、まず、NAREGI-IS が他グリッドの HPCBP コンピュータ資源の情報を収集して RENKEI-SS に提供するか、RENKEI-SS 自身が収集する必要がある。これには、第 3.3.1 節で説明した「グリッド間情報交換仕様」の欠如が問題になる。RENKEI プロジェクトでは、第 3.3.1 節の解決策のように、多くのグリッドが採用している GLUE2 スキーマで、他グリッドの情報サービスと資源情報の交換を行うものとした。ただし、NAREGI ミドルウェア全体の情報スキーマは Distributed Management Task Forth（DMTF） [51]が策定した Common Information Model（CIM） [52]に準拠しているため、RENKEI-IS に GLUE2-CIM 情報変換を行う機能を追加した。また、情報

サービス間の情報交換プロトコル仕様も決まっていないため、任意のプロトコルが使用できるように通信プロトコル部分をプラグイン形式のアーキテクチャとした。また、コンピュータ資源タイプ識別子として BES（HPCBP）を NAREGI-CIM スキーマへ追加した。RENKEI-SS の改造は、資源情報を問い合わせる Candidate Set Generator （CSG）と、その他内部でコンピュータ資源タイプを判定する必要があるサービス群、BPEL Workflow Engine（BWE）、Aggregation Service Container（ASC）、Reservation Cache Service（RCS）に BES（HPCBP）資源の判定処理を追加した。

要件④の対応について説明する。RENKEI もしくは NAREGI クライアントから RENKEI コンピュータ資源へのジョブ投入は HPCBP 仕様よりも機能が高い NAREGI 仕様のインターフェースを使用することとし、NAREGI コンピュータ資源と同じシングルサインオンを実現した。問題は HPCBP コンピュータ資源へのシングルサインオンである。NAREGI では X.509 プロキシ証明書を、HPCBP 仕様では X.509 証明書か JSDL 内に記述するユーザ名／パスワードトークンを認証クレデンシャルとして使用している。NAREGI のすべてのモジュールやサービスで X.509 証明書やユーザ名／パスワードトークンをサポートすることにすると、NAREGI のセキュリティポリシーや基本アーキテクチャにまで影響を与えるので、SS のみで対応する方法を検討する必要があった。ただし、PGI では認証クレデンシャルのデレゲーション機能が要件として挙がっているため、将来はデレゲーション方式が採用されるものとして、一時的な対応としての簡易実現方法を検討した。まず、ユーザ名／パスワードトークンのサポート方法について説明する。ユーザ名／パスワードトークンは JSDL に記述されて転送されるのが仕様であるため、SS はジョブドキュメントの JSDL を解釈することにより入手可能であり、これを使用してコンピュータ資源にログインすることができる。しかし、ユーザ名／パスワードトークン方式は、証明書方式に比べ安全性に問題がある。例えば JSDL に記述するため、たとえ通信チャネルは暗号化されて安全であっても、途中のサービスが JSDL をプレーンテキストでログすることがあるため、他人に見られる可能性がある。また、同一ユーザでもグリッドやサイトによりユーザ名が異なる場合、固定されたユーザアカウントではブローカリングできない問題がある。このため、ユーザ名／パスワードトークンしかサポートしていないコンピュータ資源はブローカリング対象から外し、ユーザが JSDL の *CandidateHosts* エレメントに明にこの資源を指定している場合のみ使用可能とする設計仕様とした。次に、X.509 証明書への対応について説明する。NAREGI における X.509 証明書は、一般ユーザがログイン不可能な NAREGI User Management Service （NAREGI-UMS） に一括して保存し、その他の NAREGI サービス群は X.509 証明書から生成した X.509 プロキシ証明書を使用する設計として、X.509 証明書本体の安全性を保っている。このため、ジョブ毎に SS へ X.509 証明書を転送することは、NAREGI のセキュリティポリシーに反することになる。そこで、UMS 同様に、一般ユーザは SS にはログインできないため、SS 上に X.509 証明書とこれに対応

するパスワードを暗号化して格納するリポジトリを設けることとした（図中の CR: Certificate Repository)。このリポジトリから X.509 証明書とパスワードを取り出すには、X.509 プロキシ証明書が必要な設計とした。SS はジョブを HPCBP コンピュータ資源に投入するとき、ジョブ投入時にデレゲートされた X.509 プロキシ証明書を使用して X.509 証明書とパスフレーズを取り出し、取り出した X.509 証明書とパスフレーズを使用して HPCBP コンピュータ資源にジョブを投入する。これによりシングルサインオンが実現される。しかし、暗号化されているとは言えリポジトリへのパスフレーズの記録は安全上問題があり、あくまでも PGI 仕様が策定されるまでの暫定的な解決方法である。

要件⑤は、NAREGI と HPCBP とでデータステージング方法が異なるために設ける必要がある要件である。HPCBP では JSDL に記述したデータステージングエレメントに従いコンピュータ資源がステージングを行う仕様であるが、NAREGI では独立したデータステージングアクティビティとして記述し SS が GridFTP の第三者転送を使用してステージングを実行する仕様になっている。このため、HPCBP コンピュータ資源が選択されたときには、この独立したデータステージングアクティビティを SS が実行しないよう制御する必要がある。具体的には SS の Execution Planning Service（EPS)、BWE、ASC においてコンピュータ資源タイプの判定を行い、BES（HPCBP）コンピュータ資源である場合はデータステージングアクティビティを実行しない制御を行う。また、要件である NAREGI コンピュータ資源や GridFTP ストレージ資源と HPCBP コンピュータ資源の間でデータステージングを可能とするには、さらに SS の File Staging Container（FSC）を改造する必要があり、ジョブドキュメントの JSDL 部にデータステージング指示があった場合、データステージングを HPCBP コンピュータ資源に委ね、SS は何もしない設計とした。

要件①は、まず上記要件②～⑤の対応が必要である。要件③を実現することにより、RENKEI、NAREGI、HPCBP のいずれのコンピュータ資源をブローカリング対象とすることができる。ジョブを投入するとき、RENKEI/NAREGI と HPCBP とでは異なる認証が必要になるが、要件④を実現すれば資源タイプ毎の認証方式で認証することが可能である。また、要件②の実現により JSDL の非互換部分はエラーが発生しないようにコンピュータ資源側で制御されるため、共通の JSDL が使用できる。また、異種コンピュータ資源間でデータステージングが必要な場合、要件⑤の実現によりファイルステージングが可能となる。要件③で説明したように、コンピュータ資源タイプによりジョブ投入インターフェースが異なるので、各資源タイプ固有のインターフェースでジョブ投入を行う必要がある。これを実現するは、NAREGI-SS が元来持っている NAREGI コンピュータ資源以外へのジョブ投入、管理機構を利用する。NAREGI-SS では、既に gLite のコンピュータ資源である lcg-CE や、Globus Toolkit Ver.4 のコンピュータ資源インターフェースである GRAM4 を実装したコンピュータ資源へのジョブ投入が可能となっている。具体的には、SS 内部に抽象的なジョブ投入、管理インターフェースを持つ Virtual Service Container（VSC）が

あり、その中に抽象インターフェースと各資源タイプ固有のインターフェースを変換するプラグイン式のアダプタを組込むことができる。また、このアダプタをコンピューティング資源タイプに従って切り替えることが出来ることが可能であり、各資源タイプ固有のインターフェースでアクセス可能な設計になっている。なお、RENKEI コンピュータ資源へのジョブ投入は、NAREGI 仕様のインターフェース（GridVM インターフェース）の使用を既定とするので、既存の NAREGI 用のクライアントとそのプラグインアダプタを使用している。また HPCBP コンピュータ資源へのアクセスは、第 3.4.2.1 節で説明した HPCBP 用クライアントとそのプラグインアダプタを用意する設計とした。

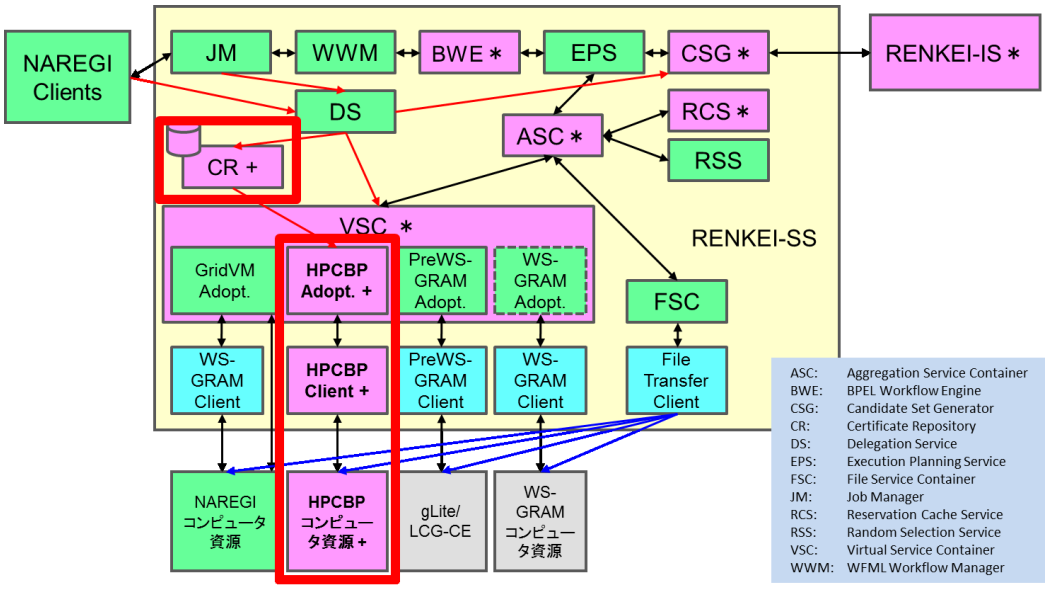


図 3-6　RENKEI-SS の構造

3.4.2.3. NAREGI コンピュータ資源の HPCBP 準拠

図 3-7 に RENKEI コンピュータ資源の構造を示す。NAREGI コンピュータ資源からの変更を ”*” 記号で、追加を “+” 記号で示す。NAREGI コンピュータ資源に HPCBP 準拠のジョブ実行サービスを追加するにあたり、満たすべき上記要件は②、⑤～⑦である。

要件②については、NAREGI、HPCBP ともコンピュータ資源に投入されるジョブ記述は OGF の Job Submission Description Language （JSDL） に準拠しており大きな問題はない。上記のように、データステージングエレメントなど若干の違いがあるが、双方のジョブ実行サービスで未サポートのエレメントは無視するように改造することにより問題を回避できる。

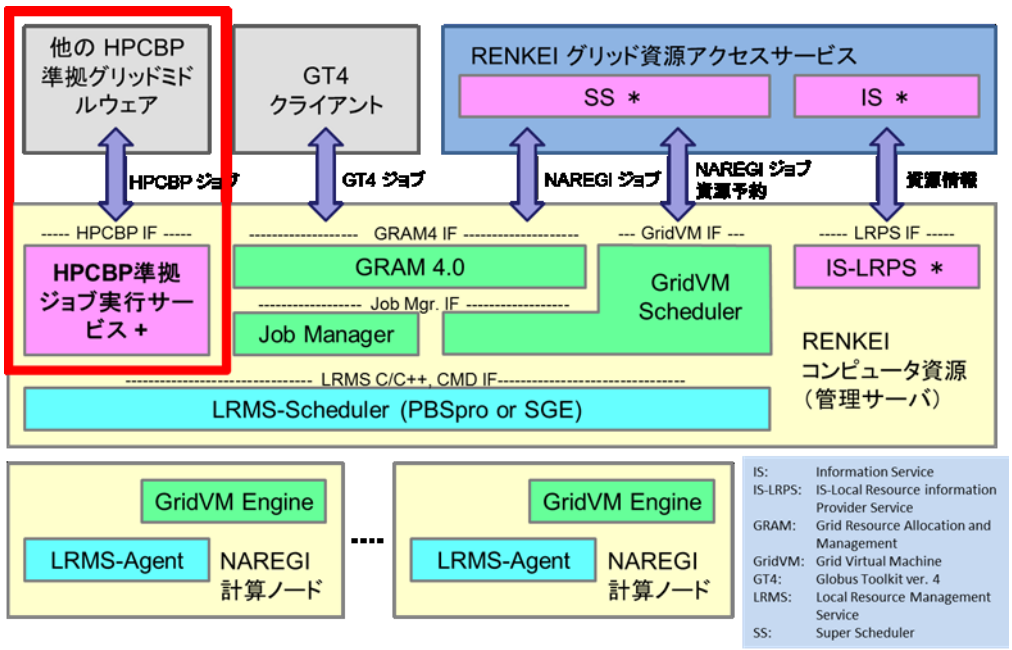


図 3-7　RENKEI コンピュータ資源の構造

要件⑤は、上記のように NAREGI ではデータステージングを SS が起動する第三者転送で実現しているのに対し、HPCBP では JSDL のデータステージングエレメントにより、ジョブ実行サービスまたはコンピュータ資源が当事者間転送を起動する。要件を満たすためのコンピュータ資源側の対応は GridVM にのみ必要であり、GridVM は JSDL にデータ転送エレメントが含まれていた場合エラーを発生せず無視する。

要件⑥は、運用している NAREGI 環境に容易に HPCBP による他のグリッドとのインターオペレーション機能を追加するための要件である。この目的はインターオペレーションために専用の計算ノードを増やすことなく、NAREGI コンピュータ資源の管理ノードのミドルウェアを RENKEI ミドルウェアに変更するだけで、簡単に他グリッドからの HPCBP 準拠インターフェースでのジョブ投入が可能とすることである。これを最小限の工数で実現するため、NAREGI コンピュータ資源としての構成を限定することにした。NAREGI コンピュータ資源は、連成計算などのコアロケーションジョブ専用の構成、コアロケーションも一般ジョブも実行可能であるがコアロケーションジョブが優先され一般ジョブは強制終了される場合がある制限付き構成、一般ジョブ専用の 3 種類の構成をとること可能である。後二者の構成では、NAREGI のジョブ実行サービスである GridVM 配下のバッチスケジューラに、GridVM とは別にエンドユーザが直接ジョブを投入することを許している。そこで、HPCBP をサポートする RENKEI コンピュータ資源では構成を後二

者のいずれかに限定することにし、HPCBP ジョブ実行サービスは GridVM とは独立にバッチスケジューラにジョブ投入をする設計とした。つまり、GridVM と HPCBP の両ジョブ実行サービスがバッチスケジューラで計算ノードを共有するアーキテクチャとした。これにより、両方のジョブ実行サービスを独立して実装することができ、ジョブ実行サービスの構造の複雑化を避けることが出来る。

要件⑦は、要件⑥で説明したように、NAREGI のジョブ実行サービスである GridVM には機能を損なうような改造をしないので、GridVM へジョブを投入すれば NAREGI の全機能を使用することができる。

3.5.　インターオペレーション研究環境の構築

開発したミドルウェアを使用して、国立情報学研究所の千葉分館にインターオペレーション研究環境を構築した。コンピュータ資源としては、NAREGI コンピュータ資源を 2 組、RENKEI コンピュータ資源、HPCBP コンピュータ資源を各 1 組で構成した。各コンピュータ資源の規模はすべて管理ノード 1 台、計算ノード 2 台である。また、他グリッドとの実証実験用に FTP サーバを HPCBP コンピュータ資源で起動している。構成を図 3-8 に示す。これらのサービスやモジュールは Xen 仮想マシン上に構築した。装置詳細を表 3-2 に示す。なお、RENKEI-SS から特定の他グリッドの HPCBP 資源にジョブ投入を可能とするため、RENKEI コンピュータ資源と HPCBP コンピュータ資源が特定の他グリッドからのジョブの受信やデータステージングを可能とするため、NII のファイヤーウォールのポートを解放した。

表 3-2　装置詳細

物理マシン	台数	2
	CPU／物理ノード	AMD Opteron 2350 (4core, 2GHz)×2
	メモリ／物理ノード	32GB
	ホスト OS	CentOS 5.5
	VM	Xen
仮想マシン	VCPU／仮想ノード	1
	メモリ／仮想ノード	4GB
	ゲスト OS	CentOS 5.5
	グリッドミドルウェア	RENKEI-final

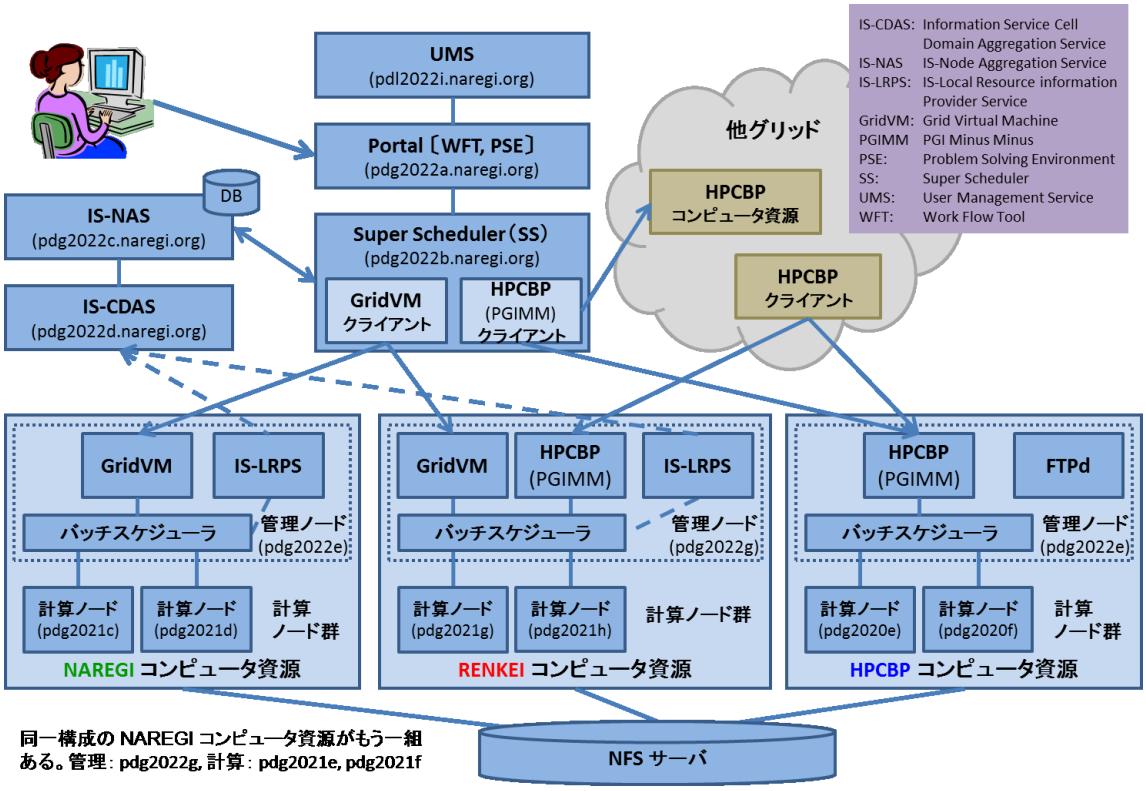


図 3-8　インターオペレーション研究環境構成図

3.6.　実証実験

3.6.1.　他グリッドからのジョブ投入実験

構築したインターオペレーション研究環境で、他のグリッドミドルウェアとの実証実験、問題点の抽出などを行った。まず、RENKEI コンピュータ資源に対し、他のグリッドの HPCBP クライアントからジョブを投入する実験を行った。実験は、英国のオクスフォード大学が開発した球体表面に配置したプラズマ電荷の最小化アプリケーションである Minem をパラメータサーベイ型のジョブとして使用した。オクスフォード大学 e-Research センターに設置した HPCBP クライアントから、日本/NII の RENKEI、ドイツ/DEISA の UNICORE、英国/UK-NGS/OMII-UK の GridSAM、ハンガリー/NorduGrid の ARC、米国/バージニア大学の GenesisII の各 HPCBP コンピュータ資源に対し、Minem に与えるパラメータだけ異なる HPCBP ジョブを投入し、結果を e-Research センターに戻し最適な結果を選択する実験であった。図 3-9 にその様子を示す。

まず準備として、各参加グリッドは Minem を各 HPCBP コンピュータ資源にインストールした。ジョブ実行の流れは以下の通りである。(1) e-Research センター： 前処理として計算の初期値を記述したファイルをコンピュータ資源数分だけ生成する。(2) e-Research

センター： ジョブドキュメントとして以下の情報を記述した JSDL を用意する。準備段階で各グリッド環境にインストールした Minem のパスを *Executable* エレメントに、データステージインとして e-Reserch センターで生成した初期値ファイルの 1 つを *DataStaging/Source/URI* エレメントに、データステージアウトとして計算結果と stdout、stderr を *DataStaging/Target/URI* に指定する。(3) e-Research センター： 各グリッドの HPCBP コンピュータ資源に対し、(2)で用意した JSDL をジョブドキュメントとして投入する。なお、指定する初期値ファイルがコンピュータ資源毎に異なるため JSDL もコンピュータ資源毎に異なる。(4) 各グリッド： 各グリッドの HPCBP コンピュータ資源は JSDL に記述された初期値ファイルを e-Research センターからデータステージインする。(5) 各グリッド： Minem を実行する。(6) 各グリッド： 実行が終了したら、結果などを e-Research センターにデータステージアウトする。(7) e-Research センター： 後処理として各グリッドから戻ってきた結果ファイルを比較して最適な結果を選択し Web にアップロードする。なお、データステージングプロトコルは FTP を使用した。動作を確認する目的の実験であったため性能評価等は行っていないが、RENKEI を含むすべてのコンピュータ資源でジョブが正常に実行され、結果が e-Research センターに正常に戻されたことを確認した。

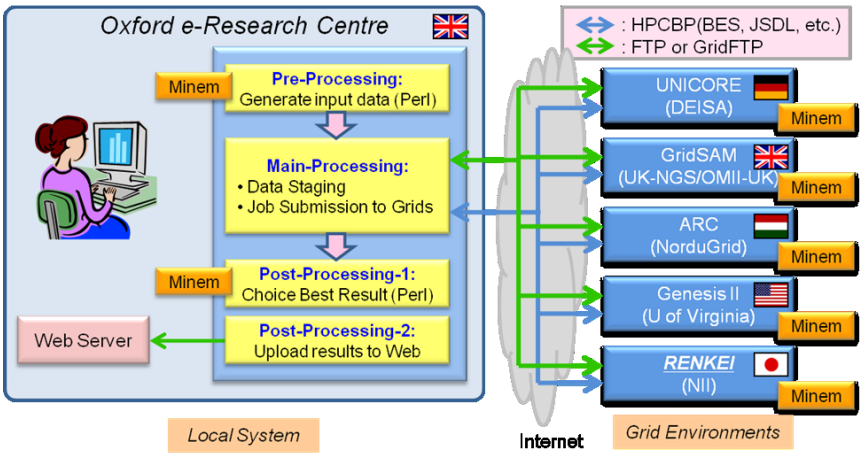


図 3-9　他グリッドから RENKEI コンピュータ資源へのジョブ投入実験

3.6.2. 他グリッドへのジョブ投入実験

次に、RENKEI-SS から他グリッドの HPCBP コンピュータ資源へのジョブ投入実験として、米国のナショナルグリッドである XSEDE の HPCBP コンピュータ資源にジョブを投入する実験を行った。XSEDE のコンピュータ資源は、San Diego Supercomputer Center（SDSC）に設置されているシステム（sierra）を使用した。第 2.2.1 節で述べたように

XSEDE はグリッド間インターオペレーションを重視した設計となっており、Enterprise Services では HPCBP を含む標準仕様に準拠したモジュール群を中心に構成されている。本実験では、上記 RENKEI ミドルウェア要件の要件③を除く①～⑤が実現されていることの確認も兼ねた。要件③については、XSEDE の情報スキーマは GLUE2 ではなく RNS [53] に準拠おり、資源情報交換ができないためブローカリング機能は使用していない。JSDL にコンピュータ資源を明に指定して実験を実施した。アプリケーションは、第 3.6.1 節の実験と同様に Minem を使用した。NAREGI コンピュータ資源で、Minem の前処理として初期値ファイルの生成を行い、この初期値ファイルを指定した JSDL で RENKEI コンピュータ資源、NAREGI コンピュータ資源、XSEDE コンピュータ資源にジョブを投入、各コンピュータ資源で Minem を実行後、結果を NAREGI コンピュータ資源に戻し後処理を実行した。データステージングプロトコルは、上記実験と同様に FTP を使用した。ジョブ実行、データステージングとも正常に実行されたことを確認した。

実験に使用した NAREGI ワークフローのスクリーンショットを図 3-10 に示す。図に示したように、ひとつのワークフロー内で NAREGI コンピュータ資源、RENKEI-HPCBP 計算資源、XSEDE-HPCBP コンピュータ資源を混在して使用している。このワークフローが正常に実行できたことにより、要件①②③④が実現できていることが確認できた。XSEDE-HPCBP コンピュータ資源に投入した JSDL（WFT 画面）を図 3-11 に示す。ここで、環境変数として Output（stdout）、Error（stderr）、DataStaging 関係の情報が記述されているが、これは NAREGI-WFT が独自に GridFTP を使用して stdout などを取得するのを防ぐための措置である。環境変数としてとして記述し、RENKEI-SS が HPCBP コンピュータ資源にジョブを投入する直前に標準的な JSDL のエレメントに変換している。また、図下部の InputOutput は NAREGI の SS によるデータステージング指定である。この指定によりデータトランスファーアクティビティが生成されるが、上記 RENKEI-SS の要件⑤対応で説明したように、HPCBP 資源に対しては SS による第三者転送を起動しない制御がされているので、この指定をしてもエラーとならず無視される。このため、この JSDL は RENKEI コンピュータ資源、XSEDE コンピュータ資源のいずれでも実行可能である。これにより要件②も実現されていることが確認できた。要件④についても、ジョブが実行でき、さらに XSEDE へのジョブ投入時にパスフレーズの入力などの手動操作が不要であったことから実現できていることが確認された。要件⑤についても NAREGI 資源からの初期値の転送、結果の NAREGI 資源の転送が正常に行われたことから実現できていることが確認された。

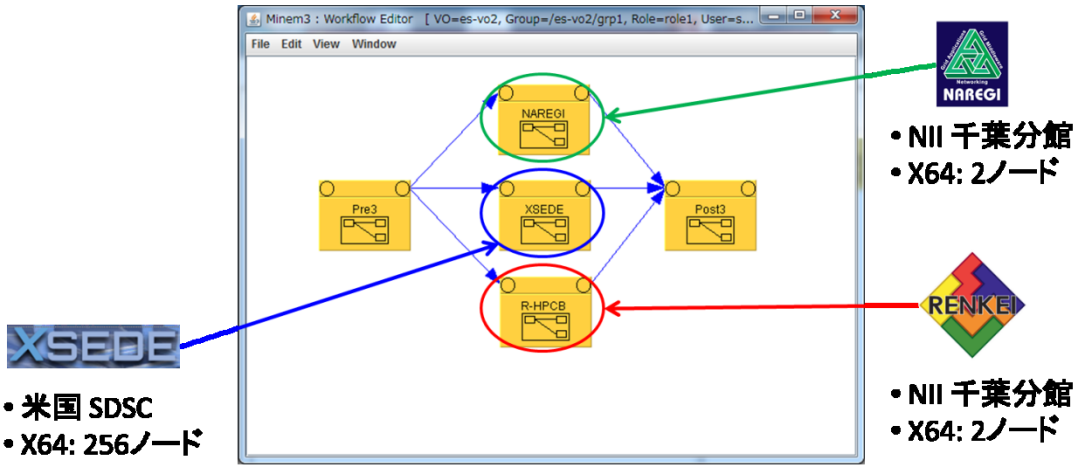


図 3-10　RENKEI-SS から他グリッドへのジョブ投入

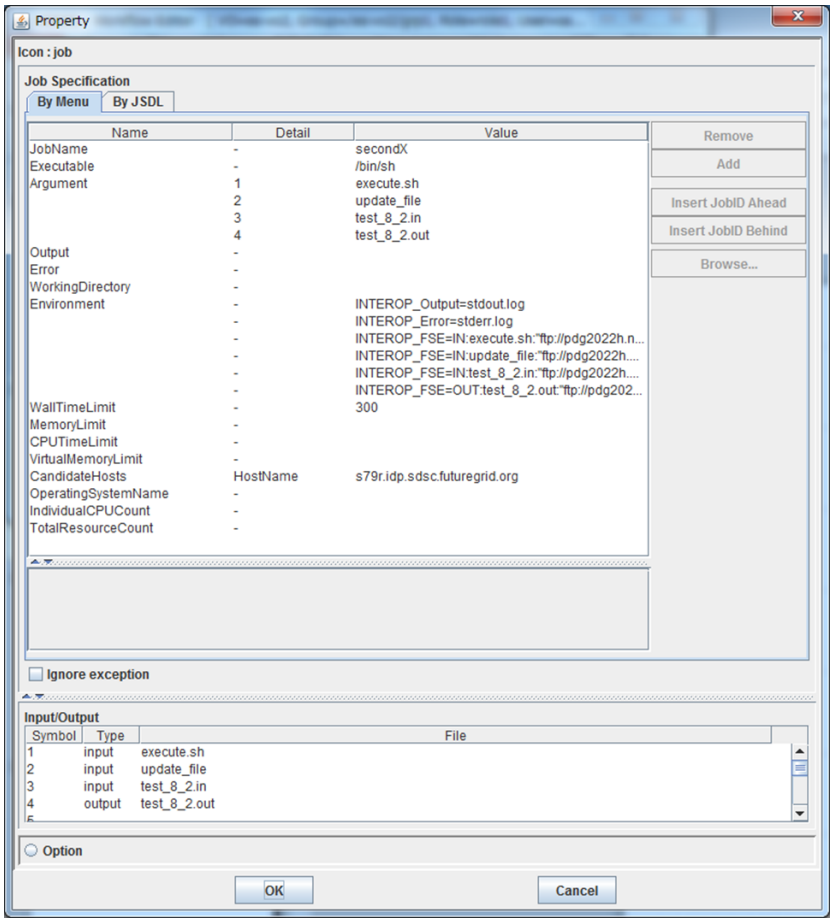


図 3-11　XSEDE-HPCBP コンピュータ資源に投入した JSDL

3.6.3. 性能測定

国際間でジョブインターオペレーションが実現できても、ジョブ投入に長時間かかるようでは使い物にならない。そこで、先の BES++クライアントを用いて、クライアントと同じネットワークセグメントにある RENKEI コンピュータ資源と、米国 SDSC の XSEDE コンピュータ資源に対し、ジョブ投入からその応答としての End Point Reference（EPR）が戻ってくるまでの時間を測定した。米国へは共有回線を使用しており、また装置性能にも左右されるため、詳細な投入時間にはあまり意味がないかもしれないが、先のインターオペレーション環境で 10 回実施した応答時間の平均は、RENKEI：0.32 秒、XSEDE：0.86 秒であった。いずれも 1 秒未満であり、ストレスを感じない実用的なジョブ投入応答時間であることが確認できた。

3.6.4. 第 3 章のまとめ

本章では、単一グリッド環境のジョブ投入ステップで必要となる機能、情報などの説明を行い、これをインターオペレーション環境に拡張するときに必要となる機能や情報を明らかにした。また、この機能や情報の要件を満たすインターオペレーションアーキテクチャの提案を行った。また、現状では HPCBP 仕様に基づくインターオペレーションを行う必要があるため、HPCBP で不足している機能や情報を明確にし、解決策を提案した。さらに、既存のミドルウェアに HPCBP 準拠機能を追加するときにも問題点があることを説明し、NAREGI への HPCBP 準拠機能追加を例に問題点を明らかにし、解決策を提案して実装を実施した。この実装を使用して今後のインターオペレーション研究のための基盤環境を構築し、諸外国のグリッドとのインターオペレーション実証実験を実施し成功した。また、米国の XSEDE コンピュータ資源へのジョブ投入の性能測定を行い、米国へのジョブ投入においても実用上問題のないレベルのジョブ投入レスポンス時間であることを確認した。

第 4 章　　性能向上手法の提案

第 1 章で述べたように、インターオペレーションは管理ドメインが異なるグリッド同士の連携であるため、従来の単一グリッド環境にはない多くの問題が発生することが予想される。例えば、他グリッドからのジョブ投入によりローカルグリッドのコンピュータ資源間のロードバランスが乱れ、実行待ち時間の増加や資源利用率の低下が発生し、またコンピュータ資源のユーザアカウント管理方法の違いから、異なる管理方法のコンピュータ資源を混在して使用する場合に、一方にとってはデータステージング回数が増加する問題が発生することが予想される。本研究ではこれら 2 つ問題について、実際に問題が発生することを実験やシミュレーションで確認し、解決方法を提案する。さらに解決策を実装したプロトタイプによる実験やシミュレーションによって提案手法が有効であること示す。

4.1.　ロードバランスの乱れ

HPCBP 仕様によるインターオペレーション環境の、任意の 1 つのグリッド環境について考える。このグリッド（以下、ローカルグリッドと記す）のクライアントから投入されたジョブは、このローカルグリッドのメタスケジューラが配下のコンピュータ資源間のロードバランスを考慮してジョブを実行する資源を決定するため、効率的なジョブ実行が可能となっている。しかし、HPCBP サービスをコンピュータ資源に実装している場合、他グリッドからのジョブはローカルグリッドのメタスケジューラを経由せず、直接コンピュータ資源に投入されるため、ローカルグリッドのコンピュータ資源間のロードバランスが乱れ、

ジョブの待ち時間の増加、資源利用効率の低下などの問題が発生することが予想される。本節では、簡単な実験でこの問題が発生することを確認し原因の検討を行う。また、ジョブ数が多くコンピュータ資源も多い実運用環境でもこの問題が発生する可能性があることを、実運用グリッドのジョブ実行ログを使用したシミュレーションにより示す。そして解決策を提案し、シミュレーションにより解決策が有効であることを示す。

4.1.1. 問題詳細

この問題発生状況を詳細に説明する。この問題は、ローカルグリッドが図 1-2 概念アーキテクチャ図の Grid-A の構造のように HPCBP サービスがコンピュータ資源に実装され、このサービスが他のグリッドからジョブを受ける場合に問題が発生することが予想される。現在の多くの HPCBP 実装は Grid-A の構造を持つ。なお、ジョブを投入する他グリッドの構造には依存しない。

まず、ローカルグリッドのクライアントが、ローカルグリッドのコンピュータ資源にジョブを投入することを考える。このときクライアントとコンピュータ資源との間にはメタスケジューラが存在しており、このメタスケジューラがローカルグリッドのコンピュータ資源間のロードバランスを考慮してジョブを実行させる資源を決定し投入する。このため、資源のロードバランスがとれた効率的なジョブ実行が期待できる。次に他グリッドがローカルグリッドのコンピュータ資源にジョブを投入することを考える。どのコンピュータ資源に投入するかは、他グリッドのメタスケジューラが決定することになる。しかし、第 3.3.1 節で説明したように、ローカルグリッドと他グリッド間ではスケジューリングに必要なコンピュータ情報の負荷情報として *TotalNumberOfActivities* しか交換できず、他グリッドのメタスケジューラはローカルグリッドのコンピュータ資源の負荷状況を把握できない。このため、他グリッドからのジョブ投入はローカルグリッドのコンピュータ資源間のロードバランスを乱してしまうことが予想される。さらに、このローカルグリッドにジョブを投入する他グリッドが複数あった場合は、これら他グリッド間での情報交換もないのでロードバランスが乱れる様子はさらに複雑になると考えられる。

4.1.2. 実験による問題発生確認

実運用を行っているインターオペレーション環境がない現在、このような問題が実際に発生するかは不明である。そこで発生可能性確認のため簡単な実験を実施した。

● **実験方法**

問題の発生状況の把握、分析をし易くするめに、専用の環境を構築して実験を行った。構成を図 4-1 に示す。ローカルグリッド（Grid-0）は NAREGI 1.1.5 をベースとした RENKEI プロトタイプで、インターオペレーションの抽象構造としては図 1-2 の Gird-A に相当する。ジョブを投入する他グリッド（Grid-1~4）は、4 グリッドともオープンソースの HPCBP 実装である BES++クライアントで構築した。BES++はクライアントのみでメタスケジューラなどのサービスは持っていないので、上位にラウンドロビンでジョブ投入資源を決定する簡易メタスケジューラを設けた。第 3.3.1 節で述べたようにローカルグリッドのクライアントやメタスケジューラはそのコンピュータ資源のワークロード、実行アクティビティ数、待ちアクティビティ数などの詳細情報は取得できるが、Grid1～4 のクライアントはローカルグリッドの各コンピュータ資源の *TotalNumberOfActivities* しか取得できない。

ローカルグリッドは 8 つの均質なコンピュータ資源を持つ。各コンピュータ資源のキューは 1 つのみであり、キューに属する計算ノードは 1 ノードだけである。つまり、1 台で HPCBP ジョブ実行サービスから計算ノードまでを兼務している。また、排他実行を設定してあり、各資源で同時に実行できるジョブは 1 つだけである。コンピュータ資源は、コア数： 8、メモリ容量： 32GB の物理マシン上に 8 つの仮想マシンを構成し、1 コンピュータ資源／1 仮想マシンとして構築した。また、NAREGI クライアント、RENKEI の各サービス、他グリッドとしての HPCBP クライアントも仮想マシン上に構築し、コンピュータ資源とは異なる同一仕様の物理マシン上で動作させた。実験環境詳細を表 4-1 に示す。

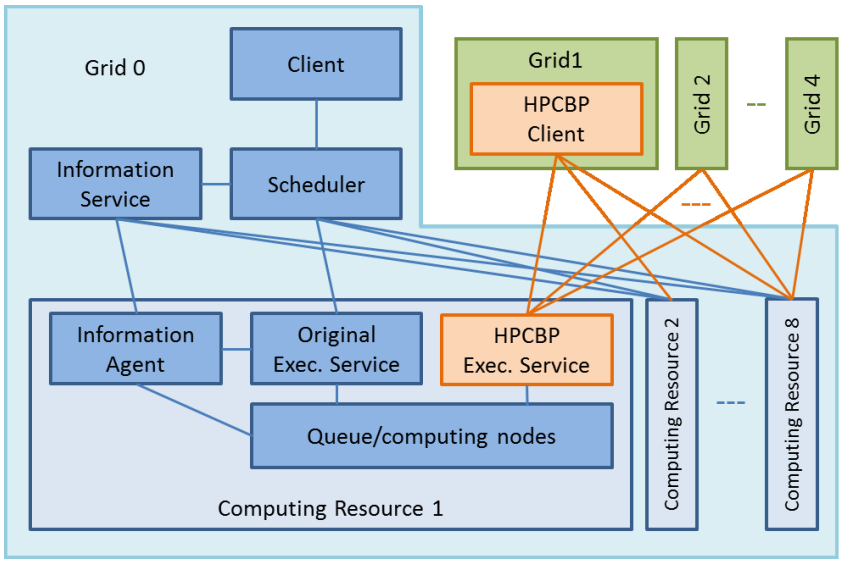


図 4-1　実験システム構成

表 4-1　実験環境詳細

物理マシン	台数	2
	CPU／物理ノード	AMD Opteron 2350 (4core, 2GHz)×2
	メモリ／物理ノード	32GB
	ホスト OS	CentOS 5.5
	VM	Xen
仮想マシン	VCPU／仮想ノード	1
	メモリ／仮想ノード	2GB
	ゲスト OS	CentOS 5.5
	グリッドミドルウェア	RENKEI HPCBP プロトタイプ 2010

● **実験方法**

入出力による実行時間の影響を受けないジョブを複数用意し、すべてをローカルグリッドのクライアントから投入する場合と、半数をローカルグリッド、残りの半数を 4 つの他グリッドから投入する場合とで、各コンピュータ資源に投入されたジョブ数、各ジョブの実行時間を比較することにより、後者ではロードバランスの乱れが発生することを確認する。

ジョブには計算が主で入出力がほとんどなく、パラメータ次第で長時間の実行が可能なニューラルネットワークのアプリケーションを使用した。入出力はジョブ開始時に約 56KB、終了時に約 118KB のファイルアクセスがあるのみであり、メモリ使用量も 16MB 程度のためページングも発生しない。このジョブを 64 個用意する。この 64 個のジョブはすべて同一入力パラメータ、同一初期値（乱数決定分も含む）を使用している。実行時間は約 560 秒である。ローカルグリッドと他グリッドで分散してジョブを投入する場合、ローカルグリッドからは半数の 32 ジョブ、他グリッドからは残りの 32 個をさらに分散させ 4 つのグリッドから 14、8、6、4 ジョブずつ投入することとした。この理由は、ローカルグリッドからのジョブ投入は他グリッドからの投入に比べて多いと考えられ、また他のグリッド間でのローカルグリッドへのジョブ投入数にはばらつきがあると考えられることによる。なお、上記のように他グリッドはローカルグリッドの負荷を把握できないため、ラウンドロビンでジョブを投入するコンピュータ資源を決定することにした。また、複数の他グリッド間でも、ローカルグリッドへのジョブ投入に対し何も情報交換がないため、ラウンドロビンの初期値（最初にジョブを投入するコンピュータ資源）は乱数で決定することにした。また、各グリッドのジョブ投入タイミングも乱数で決定した。

● **実験結果**

上記実験方法で、ラウンドロビンの初期値を変えるなどして複数回実験した。初期値やジョブ投入タイミングを乱数で決定しているため実験毎に異なる結果となったが、その中で現象が顕著に表れた実験結果を図 4-2、図 4-3 に示す。図 4-2 は、ローカルグリッドのクライアントから 64 ジョブすべてを投入した結果である。横軸は時間（秒）、縦軸はコンピュータ資源番号､図中の細かく区切られた線が 1 つのジョブの実行状態を示す｡8 つのコンピュータ資源に均一に 8 個ずつのジョブが割り当てられ、全コンピュータ資源でほぼ同じ時刻に 8 個のジョブが終了している。最初に実行を開始したジョブの開始時刻から最後に実行を終了したジョブの終了時刻までの時間は 4,775 秒であった。

図 4-3 は、ローカルグリッドから 32 ジョブ、4 つの他グリッドからそれぞれ 14、8、6、4 ジョブ投入した結果である。線色の違いは投入したグリッドの違いを示している。赤色がローカルグリッドから投入されたジョブであり、その他の色は他グリッドから投入されたジョブを示す。結果から分かるように、各コンピュータ資源で実行されるジョブ数にばらつきがありロードバランスが乱れていることがわかる。最初に実行を開始したジョブの開始時刻から最後に実行を終了したジョブの終了時刻までの時間は 5,661 秒であり、ローカルグリッドからのみジョブ投入した場合に比べ 18.6%ほど多く時間を要している。

なおいずれの実験でも、すべてのコンピュータ資源においてジョブ投入が他のジョブの実行中に発生しているため、資源がジョブ投入待ちになることはなかった。

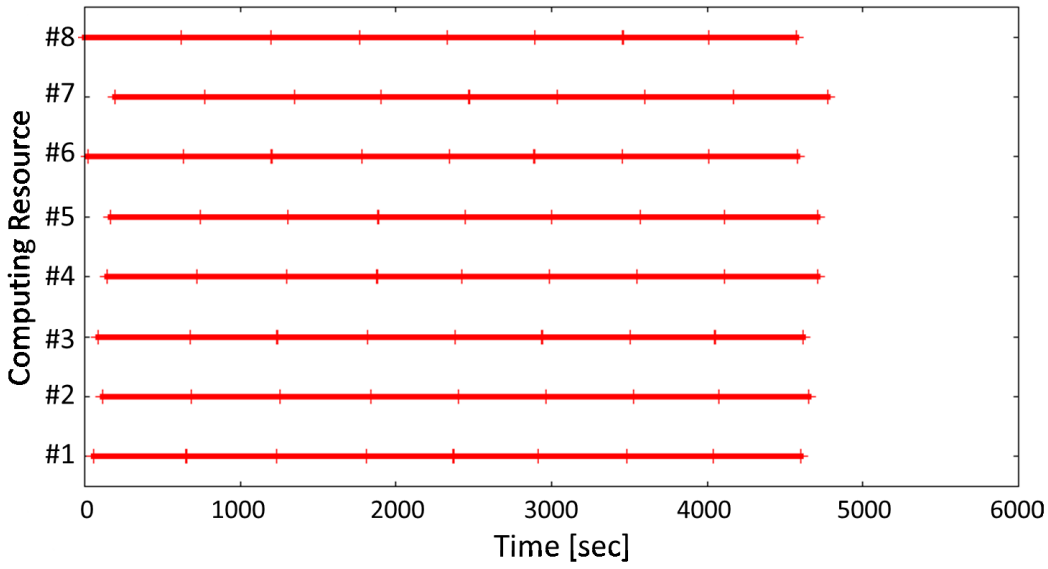


図 4-2　Grid-0 からのみジョブを投入した場合（従来の単一グリッド環境の相当）

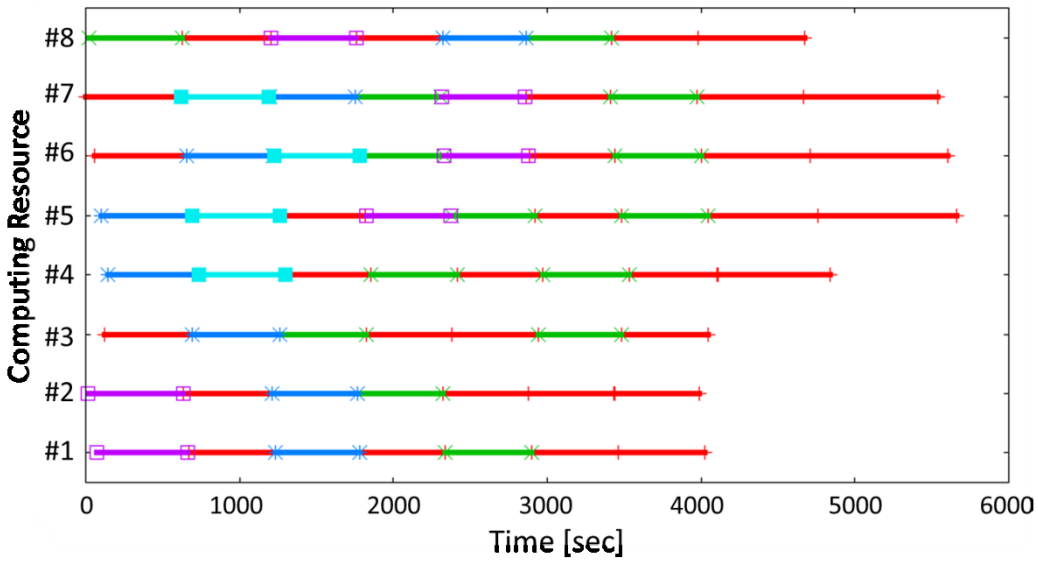


図 4-3　Grid-0~5 からジョブを投入した場合（インターオペレーション環境）

4.1.3. 考察

実験結果を考察する。複数のグリッドからのジョブ投入結果（図 4-3）では、各コンピュータ資源の実行ジョブ数が均一でなく、これが全ジョブが終了する時間の長期化をまねいている。この原因を考察する。他グリッドは投入するコンピュータ資源をラウンドロビンで決定してワークロードの分散化を図っているため、1 つの他グリッドだけに閉じればロードバランスは取れているはずである。しかし、他グリッドが複数ある場合、グリッド間でジョブ投入に関する情報交換がなく他グリッドからのジョブ投入状況がわからないため、例えば極端な例としてラウンドロビンの初期値が同じでほぼ同じ時刻にジョブを投入した場合、各グリッドの投入ジョブ数の違いからロードバランスに偏りが生じる。このような場合、もしローカルグリッドのメタスケジューラがこの偏りを検出できれば、ローカルグリッドのジョブの投入先を制御することで偏りを低減させることができる。しかし、図 4-3 の結果を見ると、各資源で 8 ジョブずつ実行されるべきところが、資源 1～3 は各 7 ジョブ、資源 5～7 は各 9 ジョブと均一になっていない。また、資源 5～7 の最後の 2 ジョブは、いずれもローカルグリッドから投入されたジョブであり、ローカルグリッドのメタスケジューラのロードバランス制御がうまく働けば、資源 5～7 の最後の各 1 ジョブは資源 1～3 に投入されたはずである。これが出来ていないのは、ローカルグリッドのジョブスケジューリングと他グリッドからのジョブ投入タイミングの間に競合状態が発生している可能性が考えられる。一般的に複数の制御主体（本件の場合は各グリッドのメタスケジューラ）が資源を共有するとき、両方からの要求がほぼ同時刻に発生した場合に競合問題が発生すること

が多い。そこで、ローカルグリッドのジョブに対するジョブスケジューリングと他グリッドからのジョブ投入がほぼ同時刻に発生したときの動作について検討した。

図 4-4 に、他グリッドからジョブ A、ジョブ B が、そしてローカルグリッドからジョブ L が立て続けに投入されたときのタイムチャートを示す。ジョブ A、B はそれぞれ時刻 t_{csa}、t_{csb} に同一のコンピュータ資源に直接投入され、ジョブ L は時刻 t_{ssl} にメタスケジューラに投入されたとする。まずジョブ A に対するローカルグリッドのメタスケジューラの動作を説明する。情報サービスエージェントが、ローカルグリッドのコンピュータ資源にジョブ A が時刻 t_{csa} に投入されたことを知ることができるのは、このコンピュータ資源の BES が、ジョブ A の認証や JSDL の検証などのジョブの正当性確認、ジョブ ID の割当てを実施した後の時刻 t_{cqa} である。この正当性確認とジョブ ID の割当てに要する時間を d_c とする。情報サービスエージェントはこれを直ちに上位の情報サービスには通知しない。ジョブ投入毎では通知頻度が高すぎ、ネットワークトラフィックの増大や情報サービスの負荷増大となるためである。情報サービスエージェントは、一般に特定の間隔 d_{ua} ごとに、ジョブ数や待ち状態ジョブ数など情報サービスに提供するための資源情報を更新する。また、情報サービスエージェントはコンピュータ資源と同数あるため、これらが各自勝手なタイミングで報告していたのでは、ネットワーク負荷や情報サービスの負荷の集中が起こる可能性がある。そこで情報サービスは、各情報サービスエージェントからの情報取得が時間的に平均化するよう調整している。たとえば、情報サービスエージェント毎に異なるタイミングで、同じ間隔でポーリングするなどである。このポーリング間隔を d_{ui} とする。これら d_c、d_{ua}、d_{ui} による遅延のため、メタスケジューラがジョブ A の投入を知ることが出来るのは投入時刻 t_{csa} から d_{da} 時間後の時刻 t_{da} である。しかし、この時点ではまだジョブ L の資源ブローカリングのための資源情報問い合わせ t_{sil} の前であるため、メタスケジューラはジョブ A を考慮してジョブ L の資源を選択することができる。この状況ではロードバランス問題は発生しない。次にジョブ B について検討する。ジョブ B の投入時刻を t_{csb}、この投入を情報サービスエージェントが知ることができる時刻を t_{cqb} とすると、メタスケジューラがジョブ B の投入を知ることができるのは、投入時刻 t_{csb} から d_{db} 時間後の時刻 t_{db} である。このとき、ローカルグリッドからメタスケジューラへのジョブ L の投入は、ジョブ B の投入より後であるにも関わらず、d_c、d_{ua}、d_{ui} による遅延のため、メタスケジューラがジョブ B 投入を認識可能になるのは t_{db} であり、資源情報問い合わせ t_{sil} より遅れるため、ジョブ L の資源決定にジョブ B を考慮できない。このため、ジョブ L とジョブ B が同じコンピュータ資源に投入されと、ロードバランスが乱れる現象が発生する。一般に d_c は秒単位の遅延であるが、d_{ua} はバッチスケジューラの制限により分単位にすることを求められることもある。また d_{ui} は、コンピュータ資源が多数ある場合は、d_{ua} よりもかなり長い間隔に設定されることも珍しくない。

これは、実験環境である RENKEI-SS、RENKEI-IS の情報交換タイミングを基に検討し

たが、他のグリッドミドルウェアでも RENKEI-IS のようなグリッド内の情報を収集、提供する統合的な情報サービスを持つ場合がほとんどであり、同様な遅延を持っている。

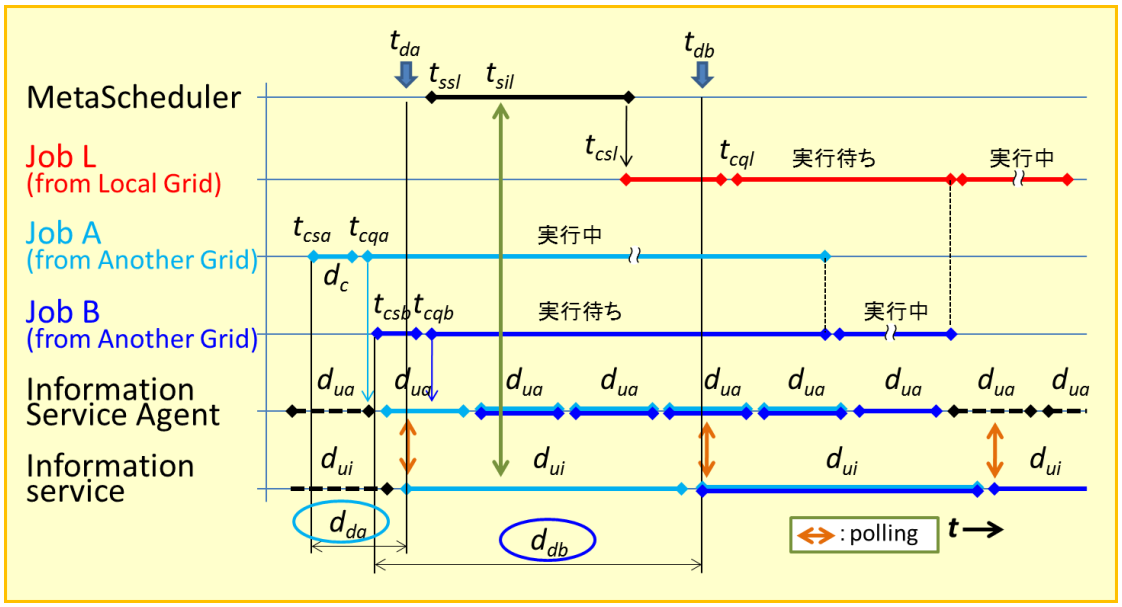


図 4-4　ジョブスケジューリングタイミング

4.1.4. シミュレーションによる問題発生の確認

上記実験は、現象を発生し易く、また現象を理解し易くするために、均一のジョブ、資源の整数倍のジョブ数、均一の 1 ノード構成のコンピュータ資源といった、実運用とはかけ離れた状況での実験であった。実際の環境ではコンピュータ資源構成も様々、ジョブ数や各ジョブの実行時間も様々である。そこで、実際の運用環境で問題が発生する可能性を調査するため、実運用のグリッドのジョブ実行ログを使用してシミュレーションを実施した。ただし、現時点においてインターオペレーションを実施しているグリッド環境はないため、公開されている複数の単一グリッドのジョブ実行ログを使用し、記録されているジョブの投入タイミング、要求資源数を基に、各グリッドのジョブを一つのローカルグリッドに投入するシミュレーションを実施した。

シミュレーションに使用したログは、The Grid Workloads Archive [54]で公開されている、DAS-2、Grid5000、NorduGrid、AuverGrid のログである。この中の 1 つをローカルグリッドに設定し、各グリッドのメタスケジューラが各グリッドのログに記録されているジョブの資源要求量に従い、記録されているタイミングで、ローカルグリッドのコンピュータ資源にジョブを投入するシミュレーションである。これらアカデミック環境の計算機使

用には繁忙期と閑散期があるため、共通の期間のログを抽出してシミュレーションを行った。またログには、実行キュー名、各ジョブの使用プロセッサ数は記録されているが、コンピュータ資源の構成情報はないため、各コンピュータ資源は 1 つのキューしか持たないものと仮定し、各キューに属するプロセッサ数は各キューで実行されたジョブが同時に使用したプロセッサの最大数を資源構成（プロセッサ数）としてシミュレーションを行った。このコンピュータ資源構成の概要を表 4-2 に、記録されているジョブの概要を表 4-3 に示す。

表 4-2　コンピュータ資源構成の概要

Grid	No. of Queues	No. of Proc.	Max No. of Proc. Of Resource	Min No. of Proc. Of Resource
DAS-2	6	179	69	4
Grid5000	34	4,173	396	1
NorduGrid	67	12,333	2,141	1
AuverGrid	53	1,362	191	1

表 4-3　記録されているジョブの概要

Grid	No. of Jobs	Max No. of Required Proc. in a Job	Min No. of Required Proc. in a Job	Max No. of Required Proc. in a time
DAS-2	328,037	66	1	7,109
Grid5000	134,288	210	1	
NorduGrid	178,369	50	1	
AuverGrid	88,745	1	1	

シミュレーションを行う前に、まずこれら 4 つのログに上記のような資源情報更新期間に複数ジョブが投入され、競合する状態が発生しているかを確認した。図 4-5 に結果を示す。横軸はジョブ更新間隔（秒）、縦軸はシミュレーション全期間における情報更新間隔内に複数のジョブが投入される回数数の累積である。いずれの情報更新間隔でも、複数のジョブが投入される回数が零ではないので、これらログを使用すれば問題が発生する可能性があることを確認した。また、1 回の更新期間内に同じグリッドから複数のジョブが投入された場合、ジョブ投入元のグリッドのメタスケジューラがラウンドロビンで別のコンピュータ資源にジョブを投入する。このため、このような状況は 1 回とカウントすると、情報更

新期間が長くなると競合状態が少なくなる。また、更新間隔が短すぎるとジョブ投入が同時期に発生する確率は小さくなるため、競合は少なくなる。これを図 4-6 に示す。結論として、これらのログはロードバランスの乱れ問題のシミュレーションに使用できると判断した。

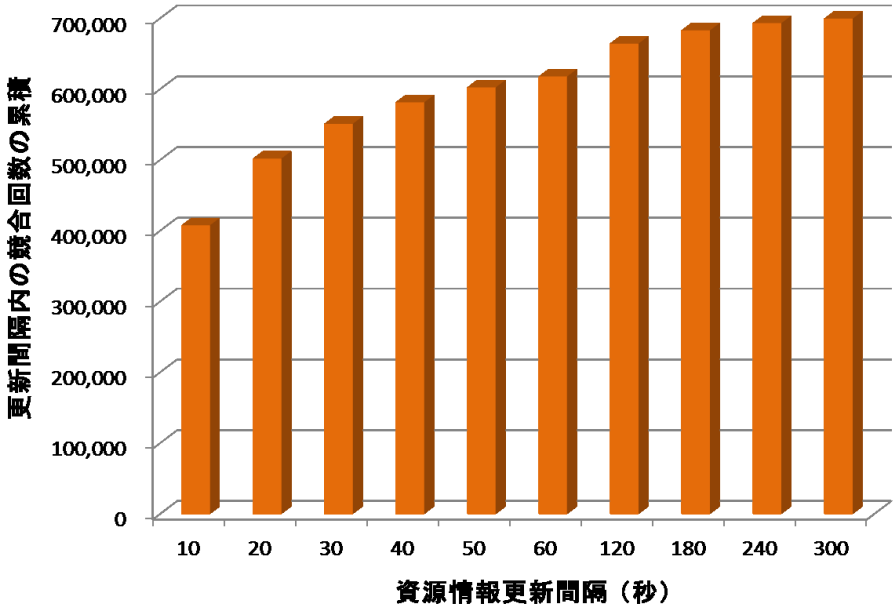


図 4-5　シミュレーションデータにおける競合状態の累積（ジョブ単位）

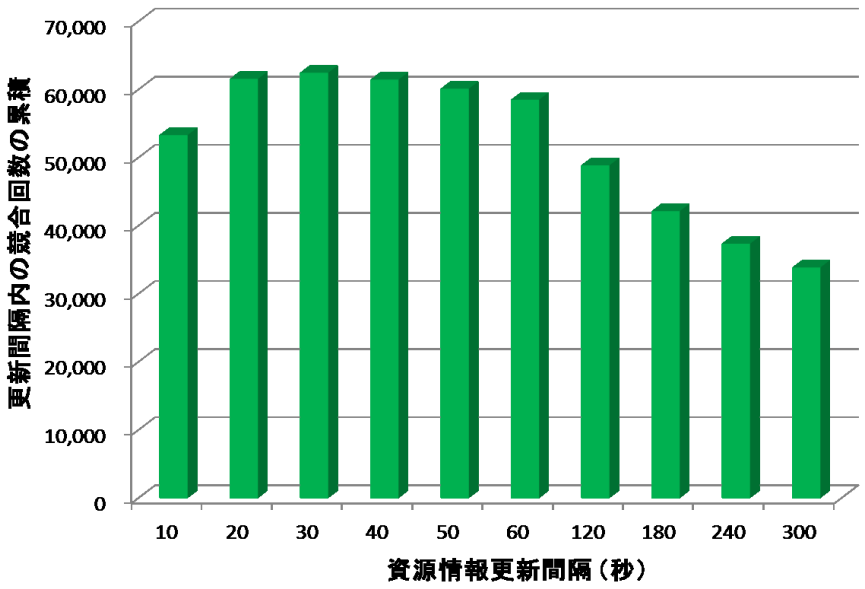


図 4-6　シミュレーションデータにおける競合状態の累積（グリッド単位）

次に、これらグリッドのローカルグリッドとしての適応性について調査した。まず、すべてのログのすべてのジョブを各グリッドのコンピュータ資源で実行できるかを調査した。要求プロセッサ数がローカルグリッドのコンピュータ資源の最大プロセッサ数より大きい場合は投入できないためである。表 4-3 からジョブが要求するプロセッサ数の最大は 210 であり、DAS-2 と AuverGrid の資源の最大プロセッサ数を上回っているため、このジョブはこれら 2 つのグリッド環境では実行できない。そこで、ローカルグリッドとして Grid5000 と NorduGrid のコンピュータ資源環境を想定してシミュレーションを実施した。また、これら 2 つのグリッド環境と全ジョブの投入タイミングについて調査したところ、NorduGrid をローカルグリッドとしてシミュレートした場合、資源が豊富にあるためロードバランスをうまくとれば、すべてのジョブが実行待ちなしに実行されることが判明した。また、Grid5000 は資源があまり豊富ではなく、どんなにうまくスケジューリングしても実行待ちのジョブをなくすことは出来ないことが判明した。このような状況は、現実に起こりうるため、この 2 つの環境をローカルグリッドとしてシミュレーションを実施した。

- **シミュレーション結果**

ログに記録されている、資源数、ジョブ数とも実験で使用したものより大幅に多いため、上記実験と同様な評価はできない。シミュレーションの目的が、クリティカルセクションにおけるジョブ投入の競合により、コンピュータ資源間のロードバランスが乱れ、結果としてジョブの実行待ちが発生し、ジョブの終了時間が遅くなることを示すことであるため、実行待ちになるジョブ数と平均実行待ち時間で評価することにした。図 4-7 に実行待ちジョブ数の累積、図 4-8 に平均実行待ち時間（秒）を示す。実行待ちジョブ数の累積は、更新間隔毎の全ログ期間における実行待ちジョブ数の累積であり、全更新間隔を通じた累積ではない。これらの結果は、最初にジョブを投入する資源と最初の情報更新時刻などランダムに決定するパラメータを変えて 5 回シミュレーションを実施した平均である。

これらの結果から、資源が十分にあるかないかにかかわらず、資源更新間隔が 10 秒でもあると実行待ちとなるジョブが発生し、最終ジョブの終了時刻が遅くなることが確認された。ただし、資源が十分ある場合はない場合に比べ平均ウェイト時間は短い。しかし、たとえ現在は資源が十分ある場合でも、ユーザやジョブ数の増加により資源が不十分になるため、この問題を解決する必要がある。

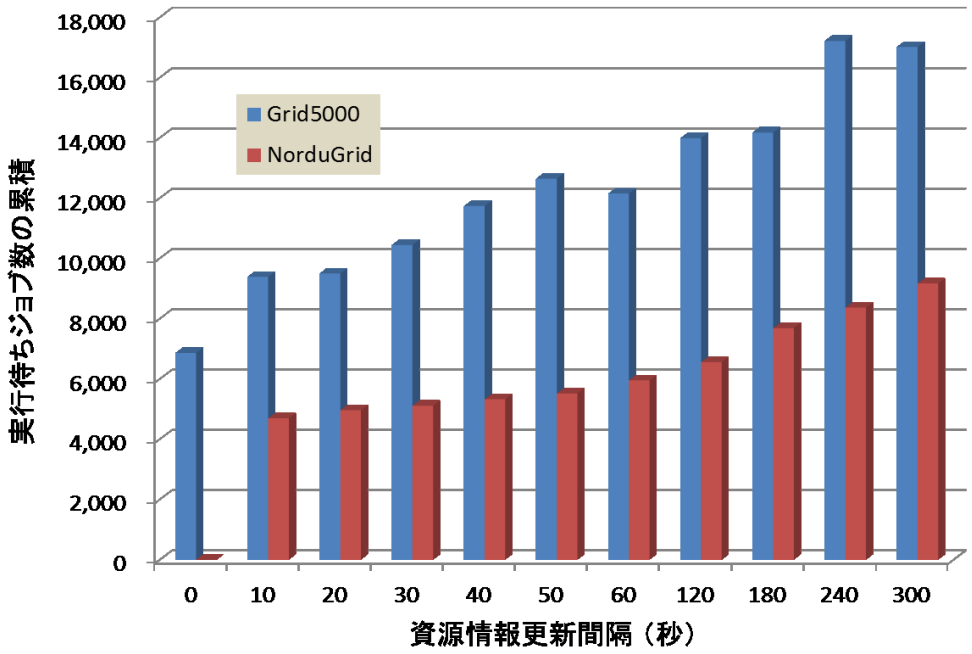


図 4-7　実行待ちジョブ数の累積

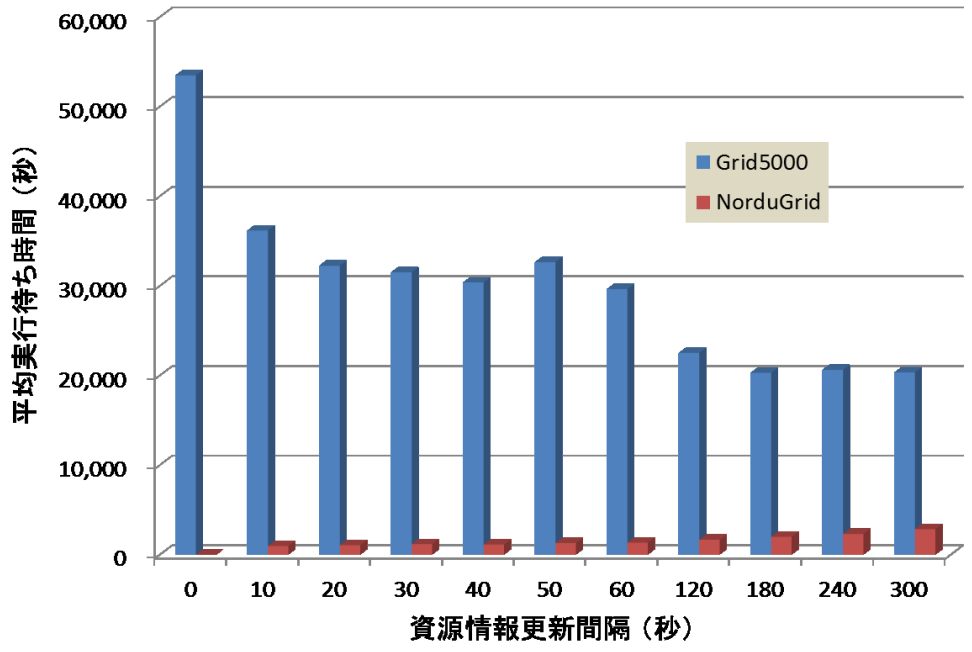


図 4-8　平均実行待ち時間（秒）

4.1.5. 解決策と評価

第 4.1.3 節の考察で示したように、本課題の原因は、他グリッドからのジョブ投入をメタスケジューラが認識するのが遅れることにある（上記 d_{db}）。そこで、これを低減するためのアーキテクチャを 3 種類検討した。以下に説明する。

- **検討 1： 情報更新遅延を低減するアーキテクチャ**

まず現状のアーキテクチャを大きく変更せずに、遅れの原因である、ジョブの正当性の確認、情報サービスエージェントの情報更新間隔、情報サービスの情報更新間隔をどこまで短縮できるかを検討した。まずジョブの正当性の確認（図 4-4 の d_c）であるが、これはユーザ認証やジョブ記述に誤りがないかの確認であり、ジョブ実行の必須作業のひとつである。チューニングして処理時間短くすることは可能でも 0 にすることはできない。また、現状でも 0.5 秒以下のためこの部分の低減努力をしても効果はあまり大きくない。次に情報サービスエージェントの情報更新間隔(同図 d_{ua})によるジョブ投入検出の遅れを検討する。情報サービスエージェントは、実行ジョブ数、実行待ちジョブ数などの負荷情報を提供するために、下位のバッチスケジューラから情報を収集する。しかし、一般にこの情報収集を頻繁に行うとバッチスケジューラへの負荷が高くなるため、バッチスケジューラによっては問い合わせ間隔を 1 分以上とするよう求めるものもある。したがって、従来の構造ではこれを短くすることはできない。そこで、HPCBP サービスが他グリッドからジョブを受信したときに、メタスケジューラにジョブ投入イベントを直接通知する方法を検討した。この構造を図 4-9 に示す。図中の青い破線がこの直接通知する仕組みである。これには、バッチスケジューラを経由していないため、直ちに実行が始まるのか、実行待ちになるのか分からないといった問題がある。しかし、次の情報サービスによる情報更新では正しい状態が報告されるので、大きな問題とはならないと考えている。HPCBP サービスからから直接メタスケジューラに報告する仕組みのため、情報遅延（図 4-4 の d_{ui}）は 1 秒以下と考えられ、ジョブの正当性の確認、情報サービスエージェントの通知時間の遅延を合わせても 1 秒程度になることが見込まれる。遅延が完全には 0 にならないので、まだ問題が発生する可能性は残っているが、従来の分単位に比べ 1 秒以内に複数のジョブの競合が発生する可能性はかなり低くなると考えられ、問題はかなり改善されると考えられる。しかし、情報の流れが複雑で実装が複雑になること、ジョブ毎の報告によるメタスケジューラの負荷の増加やネットワークトラフィックの増加してしまうなどの問題がある。

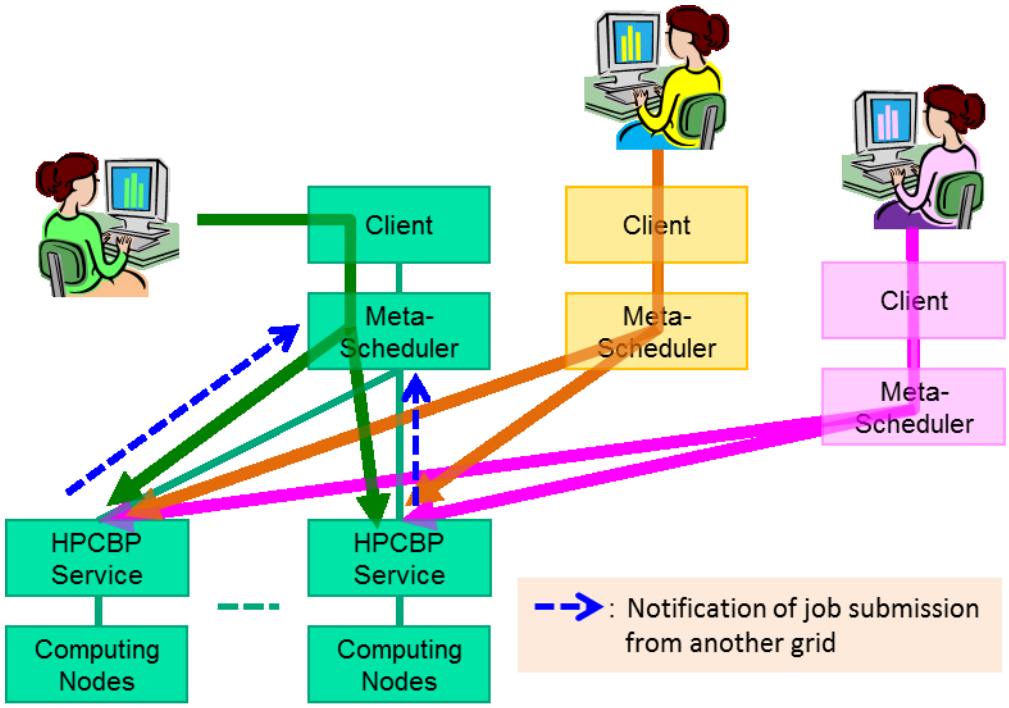


図 4-9　情報更新遅延を低減

● **検討 2：　メタスケジューラの HPCBP 準拠**

本課題が発生する根本原因は、他グリッドがローカルグリッドのメタスケジューラを経由しないで、ローカルグリッドのコンピュータ資源に直接にジョブを投入することにある。メタスケジューラが、ローカルグリッドから投入されたジョブの投入資源を決定している間に、他グリッドが偶然同じコンピュータ資源にジョブを投入することにより発生する。そこで、メタスケジューラに HPCBP 準拠のインターフェースを追加し、他グリッドからのジョブも必ずローカルグリッドのメタスケジューラを経由して、コンピュータ資源にジョブを投入することを検討する。構成を図 4-10 に示す。この方法によれば、他のグリッドからのジョブもメタスケジューラがロードバランスをとってコンピュータ資源に投入することになるので、ロードバランスの乱れ問題は完全に解消する。しかしこの構造の問題は、他グリッドからのジョブは他グリッドのメタスケジューラとローカルグリッドのメタスケジューラの両方を通ることである。先に述べたように、メタスケジューラの処理には時間を要する場合が多く、RENKEI-SS では約 40 秒要している（上記インターオペレーション研究環境の場合)。他グリッドのメタスケジューラでも同様な処理時間が必要と思われ、実行時間の短いジョブを多数処理する High Throughput Computing（HTC）ジョブの場合、そのジョブ投入レスポンスの遅さが問題となる。参考までに、先のシミュレーションで使用した実運用グリッドのジョブの実行時間を表 4-4 に示す。全ジョブの 48%が

100 秒未満のジョブである。例えば他グリッドのメタスケジューラの処理時間も RENKEI と同程度の約 40 秒かかるとして、100 秒のジョブを実行ために最低でも 80 秒（40+40 秒）も投入のために時間がかかることはターンアラウンド時間の観点から問題である。そこで、他グリッドからのジョブがコンピュータ資源に投入される前にメタスケジューラが認識することが可能でありながら、ジョブ投入レスポンス時間を短くする方法を検討する必要がある。

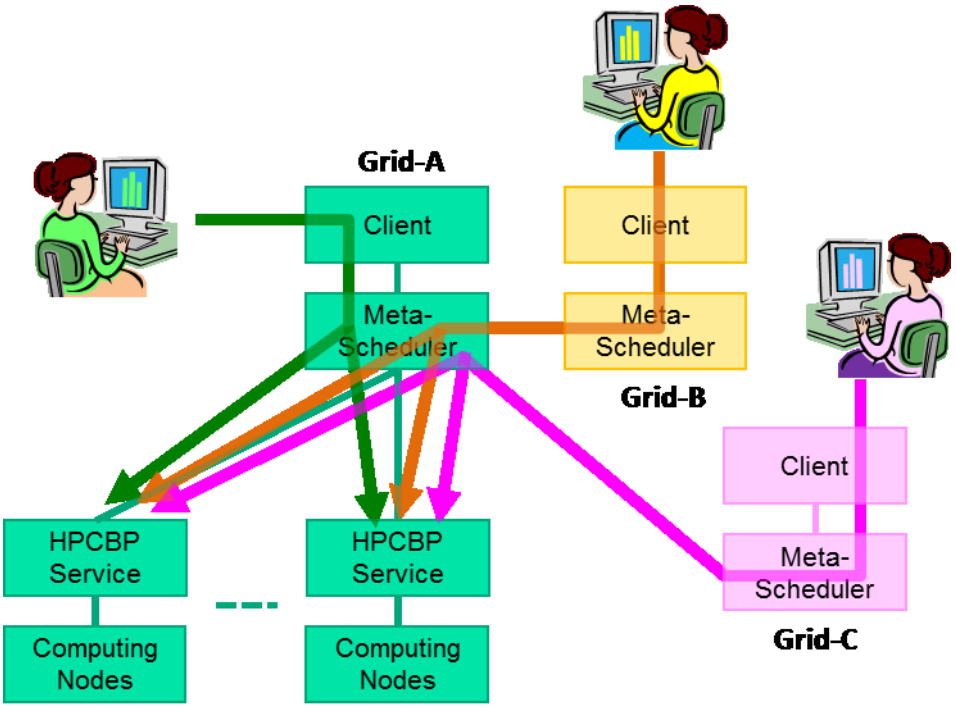


図 4-10　メタスケジューラに HPCBP 準拠インターフェースを実装

表 4-4　ジョブの実行時間

実行時間（秒）	DAS-2	Grid5000	NorduGrid	AuverGrid	Total
1 ~	56,841	13,194	2,183	11,066	83,284
10 ~	153,928	78,803	17,383	18,694	268,808
100 ~	114,238	28,421	24,312	10,116	177,087
1,000 ~	2,450	8,851	44,856	23,085	79,242
10,000 ~	557	4,030	70,341	23,180	98,108
100,000 ~	23	978	18,974	2,604	22,579
1,000,000 ~	0	11	320	0	331

- **検討 3： インターオペレーションジョブゲートウェイ**

検討 2 で述べたように、メタスケジューラで他グリッドからのジョブを受信すればロードバランスの問題は解決するが、ジョブ投入レスポンス時間の問題が発生する。そこで、このジョブ投入レスポンス時間を低減する方法を検討した。ジョブ投入時のメタスケジューラの処理は、ユーザ認証、ジョブ管理情報の生成と管理、ワークフローエンジンによる流れ制御、JSDL 解析、ブローカリング、スケジューリング、抽象 JSDL の具現化、コンピュータ資源へのジョブ投入などからなっている。ここで、他グリッドから投入されたジョブを考えると、このジョブは既に他グリッドのメタスケジューラによってブローカリングされ JSDL も具現化されたジョブである。このため他グリッドからのジョブの処理として、上記ステップのうち、ユーザ認証、ジョブ管理情報の生成と管理、スケジューリング情報テーブルの更新、コンピュータ資源へのジョブ投入だけあればよく、その他のステップは不要である。メタスケジューラにおけるこれらの不要ステップに要する時間は大きく、例えば RENKEI ではメタスケジューラの処理時間の 3/4 以上を占めている。これらの処理をスキップできれば、ジョブレスポンス時間は大幅に改善される。そこで、上記 4 つの必要なステップの機能を持つ新たなサービスを提案する。これをインターオペレーションジョブゲートウェイ（以下 IJGW と略）と呼ぶ。構造を図 4-11 に示す。IJGW は他グリッドのメタスケジューラとローカルグリッドのコンピュータ資源との間で、ジョブの投入・管理、モニタの中継をおこなう。このため、IJGW は上記 4 つの機能の他に、ジョブをフォワードしたコンピュータ資源へのジョブ管理やモニタなどのリクエストと、その応答のフォワーディング機能を持つ。

IJGW に投入された他グリッドからのジョブは、まずユーザ認証が行われる。そして JSDL の *CandidateHosts* に記述されているコンピュータ資源について、メタスケジューラと共有しているスケジューリング情報テーブルの更新を行う。メタスケジューラも同時期にローカルグリッドからのジョブ投入を処理していることを考慮し、排他制御をした上で更新する。なお、この段階ではジョブは実行されていないので、実行待ちジョブとしてスケジューリング情報テーブルに記録される。この更新が正常に終了したら、スケジューリングテーブルの排他制御を解除し、ジョブをコンピュータ資源にフォワードする。BES は仕様として、ジョブを投入した BES 資源以外の BES 資源へのジョブマイグレーションが考慮されている。これを利用する。ただし、HPCBP で BES のジョブフォワードを使うには如何に認証クレデンシャルを安全な転送するかが問題となる。これはミドルウェア毎に解決方法が異なる。PGI 仕様では証明書のデレゲーション機能が要件とされているため PGI 仕様によるインターオペレーションであれば問題ない。

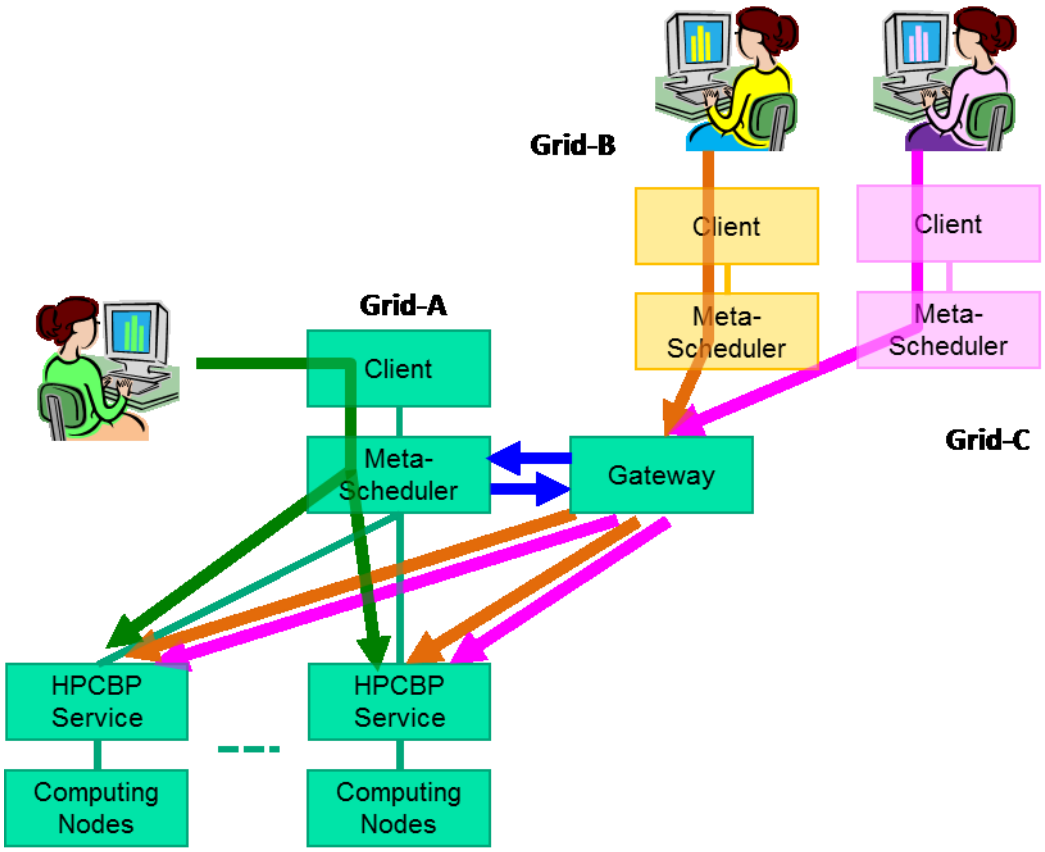


図 4-11　インターオペレーションジョブゲートウェイ

4.1.6. シミュレーション

インターオペレーションゲートウェイの有効性を確認するために、4.1.6 節と同じシミュレーションを実施した。結果を表 4-5 示す。提案手法では、情報サービスの情報更新間隔に依存せず、実行待ちジョブ数の累積、平均実行待ち時間とも一定となる。このため、RENKEI の情報サービスの更新間隔である 5 分のときの結果と並べて示した。実行待ちジョブ数の累積に関しては、ジョブに対する資源が十分にある場合でもない場合でも、提案手法では待ちジョブ数が少なく有効であることが確認された。しかし、平均実行待ち時間を見ると、資源が十分でない Grid5000 では提案手法の方が長く効果がないように見える。しかし、図 4-8 の従来手法のシミュレーションの情報更新間隔が 0 秒のときの結果と同じ値であり、提案手法による問題ではない。シミュレーション結果を調査したところ、ジョブの要求資源量の問題で、長時間待たされるジョブがこの性能低下を支配しており、実行待ちジョブ数が少ないため、平均をとると待ち時間が大きく見えることが判明した。したがって、本提案の方式は有効であることが確認された。

表 4-5　インターオペレーションジョブゲートウェイシミュレーション結果

Gird	実行待ちジョブ数の累積		平均実行待ち時間（秒）	
	従来手法	提案手法	従来手法	提案手法
Grid5000	17011.0	6856.6	20311.5	53463.7
NorduGrid	9161.2	0.0	2821.3	0.0

4.1.7. 第 4.1 節のまとめ

インターオペレーション環境で発生が予想される問題のひとつとして、ローカルグリッドと他グリッド、他グリッドと他グリッドのジョブ投入がほぼ同時に発生し、ジョブ投入の検出の遅れにより、コンピュータ資源間のロードバランスが乱れることが予想される。これが現実に発生する可能性があることを実験、ならびに実運用グリッドのジョブ実行ログによるシミュレーションによって確認した。また、この原因に対する詳細な検討の結果、この遅延は情報サービスエージェントや情報サービスの資源情報更新間隔による影響によるものであり、これを低減するアーキテクチャを検討すべきとの結論を得た。この結論を基に 3 種類のアーキテクチャの検討を行い、メタスケジューラと並行して動作する軽量のジョブ投入ゲートウェイを設け、両者でジョブスケジューリングテーブルを共有することにより、他グリッドからのジョブがコンピュータ資源に投入される前にローカルグリッドのメタスケジューラがこれを検出できる、資源情報の更新を待たないアーキテクチャを提案した。また、このアーキテクチャの有効性をシミュレーションにより示した。

4.2. データステージング回数問題

第 1 章で述べたように、コンピュータ資源のユーザアカウントの管理方法には、ジョブ毎に動的に割当てる一時アカウント方式と、従来の単一コンピュータシステムのように恒久的に割当てる静的アカウント方式があり、方式の違いによりデータステージング動作に違いがある。依存性のあるジョブのデータステージングについて両方式の動作を比較すると、静的アカウント方式ではコンピュータ資源間で直接転送できるのに対し、一時アカウント方式では中間ストレージを介した転送となる。つまり一時アカウント方式では同じデータを 2 回転送することになる。データが巨大化している昨今、この 1 回多いデータ転送はジョブ実行時間の長期化、ネットワークトラフィックの増大をもたらす。この問題だけを見ると静的アカウント方式の方が優れており、静的アカウントに統一すれば良いように思えるが、この問題点以外にも両者は相反する特徴を持ち優劣がつけられない。どちらを採用するかは各グリッドミドルウェアの設計思想による。本章では両方式の特徴、データステージング問題の詳細を説明し解決策を提案する。この問題はインターオペレーションに参加するすべてのミドルウェアが対応する必要がある問題のため、解決策は現在 OGF で策定中の次世代インターオペレーション仕様 PGI の要件に沿った方法とした。また、提案方式を第 3 章で開発したグリッドミドルウェアに実装し、実証実験を実施し効果を確認したので報告する。

4.2.1. 問題定義

まず、両方式の特徴を説明する。一時アカウント方式はグリッドミドルウェアとして広く使用されている gLite などで採用されている。コンピュータ資源はジョブが投入されたとき、認証クレデンシャルによってユーザを識別し、このユーザに対してプールされているアカウントの一つを割り当てる。さらに、このジョブに対して一時的な作業領域を割り当てる。この一時的なアカウントと作業領域は、そのジョブが存在している間のみ有効である（実際には同一ユーザが同一コンピュータ資源に複数のジョブを投入した場合には同一プールアカウントが割り当てられ、一つのジョブが終了しても他方のジョブが実行中であればアカウントは残る。説明が複雑になるので、ここではジョブが 1 つの場合を考える）。プールアカウントは実際のユーザとは基本的に無関係で、ジョブ実行時に一時的な関係を持つだけである。このためユーザの増減に伴うコンピュータ資源のアカウント管理が不要であり、コンピュータ資源数が多い場合には管理コストを大幅に低減できるなどの利点がある。しかし、ジョブ実行時の一時的な関係は、割り当てられた作業領域外に不用意にファイルを残すと、後から同プールアカウントを使用する他のユーザによって参照が可能などセキュリティ上の短所がある。一方、RENKEI や NAREGI などでは、ユーザ毎に固定された静的アカウント採用し home などの固定した作業領域を持つ。これらは一時アカウン

ト環境と異なり、ジョブの実行状態とは無関係に持続的である。また、home に加えジョブ毎の作業領域を持つ場合も多く、このライフタイムはミドルウェアや環境設定により異なる。静的アカウントには、ローカル計算機システムと同レベルのセキュリティ管理ができ、またジョブの再実行時や同じコンピュータ資源上のジョブ間ではファイル転送が不要などの利点がある。一方、ユーザが増減するたびに、そのユーザにジョブ実行を許可するすべてのコンピュータ資源でアカウントを管理しなくてはならず、コンピュータ資源数が多い場合の管理コストは大きなものとなる。また、設定ミスによる複数の資源間で動作の違いやセキュリティ上の問題を引き起こしやすいといった短所がある。このように両方式には相反する特徴があり、どちらの方式を選択するかは各グリッドミドルウェアの設計思想や要件による。

これらアカウントの管理方法の違いによる、ジョブ間のデータステージング動作の違いを依存関係のある 2 つジョブを例に説明する（先行：ジョブ 1、後行：ジョブ 2)。後行のジョブ 2 の実行には、先行のジョブ 1 の結果ファイルが必要であり、ジョブのデータステージングにより引き渡されるものとする。図 4-12 に静的アカウントの例を示す。依存関係のあるジョブのため、ジョブ 2 はジョブ 1 が終了してから起動されるが、Site-A のコンピュータ資源にはジョブ 1 の計算結果が残っているので、ジョブ 2 は Site-A のコンピュータ資源からジョブ 1 の結果ファイルを直接転送（データステージイン）することができる。次に一時アカウントの場合の例を図 4-13 に示す。一時アカウントと一時作業領域はジョブ終了時に無効化されるので、ジョブ 1 はその実行中に結果ファイルを一時作業領域外に転送（データステージアウト）する必要がある。このとき、ジョブ 2 のコンピュータ資源に転送できればよいが、一般にジョブ 1 へのジョブ投入時にはジョブ 2 を実行するコンピュータ資源は決定されていないため（ブローカリングされていない)、ジョブ 2 のコンピュータ資源へは直接転送できない。また、逆にジョブ 2 からジョブ 1 のコンピュータ資源から実行結果を転送（データステージイン）しようにも、ジョブ 2 の実行時にはジョブ 1 は終了しているのでジョブ 1 のアカウントも結果も残っていない。このため、これら 2 つのジョブの間での中間結果の引き渡しには中間に一時ストレージが必要となり、静的アカウントの場合と比べファイル転送が 1 回多く必要となっている。

近年のコンピュータ資源の高性能化による計算量の飛躍的増大は、計算結果として生成されるデータの巨大化をもたらしている。この 1 回多いデータステージング時間によるジョブ実行時間の長期化や、同じデータを 2 回ステージングすることによるネットワークトラフィックの増加は無視できない問題となってきている。しかし、HPCBP 仕様ではこの問題を解決することは出来ない。

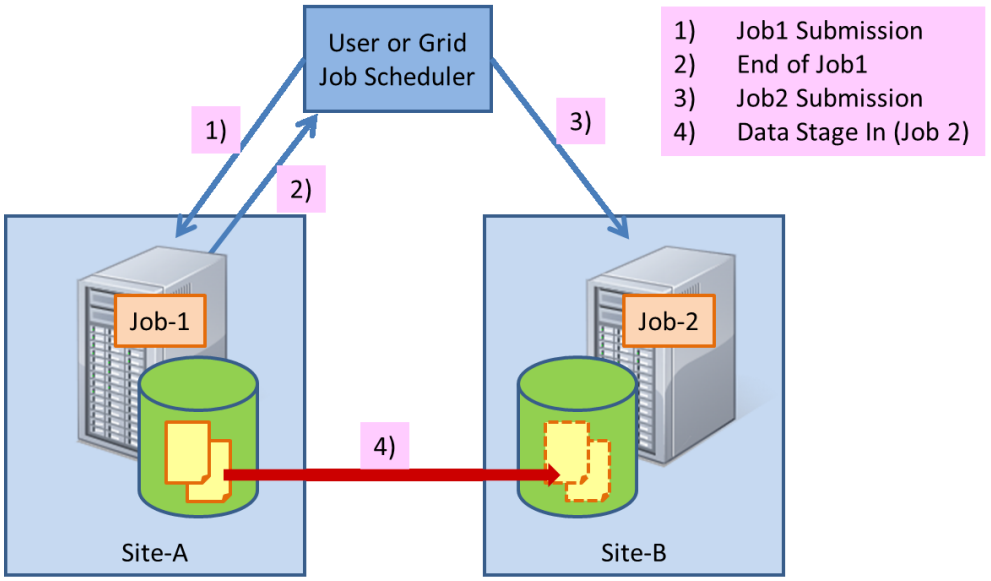


図 4-12　静的アカウント環境における依存関係のあるジョブのデータステージング

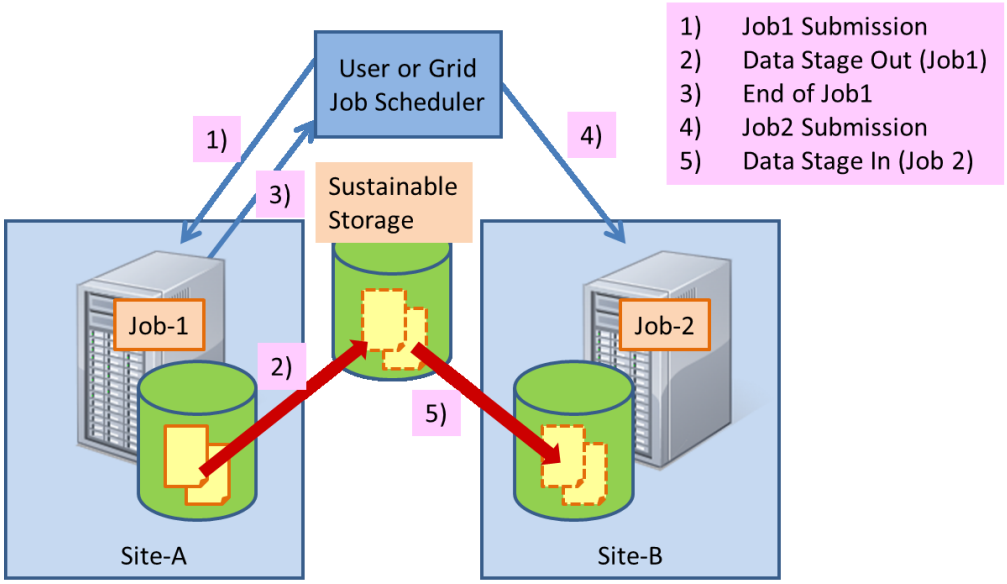


図 4-13　一時アカウント環境における依存関係のあるジョブのデータステージング

4.2.2.　提案手法

一時アカウント管理のコンピュータ資源において依存関係のあるジョブのデータステージングで中間ストレージを必要とするのは、上記の説明から分かるように、ジョブ間でア

カウントと作業領域のライフタイムの重なりがないためである。そこで、ジョブを HOLD する仕組みを設け、先行ジョブのアプリケーションがエラーなどを出し切った後に HOLD し、その間に後行ジョブを起動することによりライフタイムの重なりを作り、両ジョブ間で直接データステージングを可能とする手法を提案する。以下にその詳細を説明する。

4.2.2.1. 基本アイデア

一時アカウント方式のコンピュータ資源間で、依存関係のあるジョブ間でファイルを直接転送するのに必要な最低条件を考える。一般に依存関係のあるジョブの投入制御では、先行ジョブの処理が終了してから後行ジョブを投入する。一時アカウント方式では、ジョブが終了するとコンピュータ資源上のアカウントと作業領域は消去されるため、直接ファイル転送するには少なくとも先行ジョブが終了する前に後行ジョブを起動し、両ジョブの一時アカウントと作業領域を同時に存在させる必要がある〔必要条件 1〕。同時に存在していれば、先行ジョブの計算ステップが終了した時点で、先行ジョブが後行ジョブのコンピュータ資源へ計算結果をステージアウトするか、後行ジョブが先行ジョブのコンピュータ資源から計算結果をステージインすることが可能となる。ただし、先行ジョブの投入時には後行ジョブのコンピュータ資源が決定されていないため、実際には後行ジョブからステージインする制御になる。

従来手法では、依存関係のあるジョブは先行ジョブの終了状況によって後行ジョブの実行の可否やジョブ内容やコンピュータ資源などの変更が可能であるため、これを実現可能としなくてはならない。従来手法でこのような後行ジョブの実行条件を決定するのは、先行ジョブの計算終了状況と計算実行結果ファイルなどのステージアウト動作の結果（成否）である。これらの終了結果情報を、先行ジョブがジョブとして完全に終了する前に上位メタスケジューラに報告できなくてはならない〔必要条件 2〕。

メタスケジューラはこれらの終了結果状況の報告を受け、先行ジョブがジョブとして終了する前に後行ジョブの流れや資源を決定して後行ジョブを投入しなくてはならない。しかし、先行ジョブの計算終了状況の報告とジョブとしての完全な終了との間隔は短いため、後行ジョブが実行する前に先行ジョブは完全に終了してしまう。このため、先行ジョブの完全終了と後行ジョブ起動を同期させる機構が必要である〔必要条件 3〕。

これらの必要条件を満たすには、先行ジョブの計算やステージアウトが終了した時点で先行ジョブを一旦停止させ、計算終了状況とステージアウト終了状況の情報をメタスケジューラに報告し、メタスケジューラはこの情報を基に後行ジョブの流れや資源を決定して後行ジョブを投入すればよい。この仕組みにより両ジョブを同時に存在させ、この間にデータステージングを行うことにより、問題を解決することができる。本問題の解決方法として本方式を提案する。従来手法と提案手法の依存関係のあるジョブの流れの概要を図

4-14 に示す。従来手法では、先行ジョブの計算、ステージアウトがシーケンシャルに実行され、計算結果の中間ストレージへのステージアウトが終了した時点でメタスケジューラに終了報告する。メタスケジューラはこれを受け、後行ジョブの流れや資源を決定して後行ジョブを投入する。後行ジョブは中間ストレージからの先行ジョブの計算結果のステージインと計算をシーケンシャルに実行する。一時アカウントと一時作業領域のライフタイムは、先行ジョブはステージアウト終了まで、後行ジョブはステージインからであり重なることはない。一方提案手法では、先行ジョブの計算が終了しアプリケーションの警告やエラーが出尽くした時点で HOLD する。計算結果はステージアウトせずコンピュータ資源上に残ったままである。この時点で先行ジョブはメタスケジューラに計算終了報告を行う。メタスケジューラはこれを受け、後行ジョブの流れや資源を決定し後行ジョブを投入する。後行ジョブは直ちに先行ジョブの計算結果を先行ジョブのコンピュータ資源から直接にデータステージインする。データステージインが終了した時点でメタスケジューラに報告する。メタスケジューラはこれを受け、先行ジョブの HOLD を解除する。先行ジョブの一時アカウントと一時作業領域のライフタイムは、先行ジョブの HOLD 終了までであり、後行ジョブの一時アカウントと一時作業領域のライフタイムはステージインから始まる。HOLD とデータステージインのライフタイムが重なっているので、両ジョブの一時アカウントと一時作業領域は有効であり、この間に両ジョブのコンピュータ資源の間で直接データステージングが可能となる。

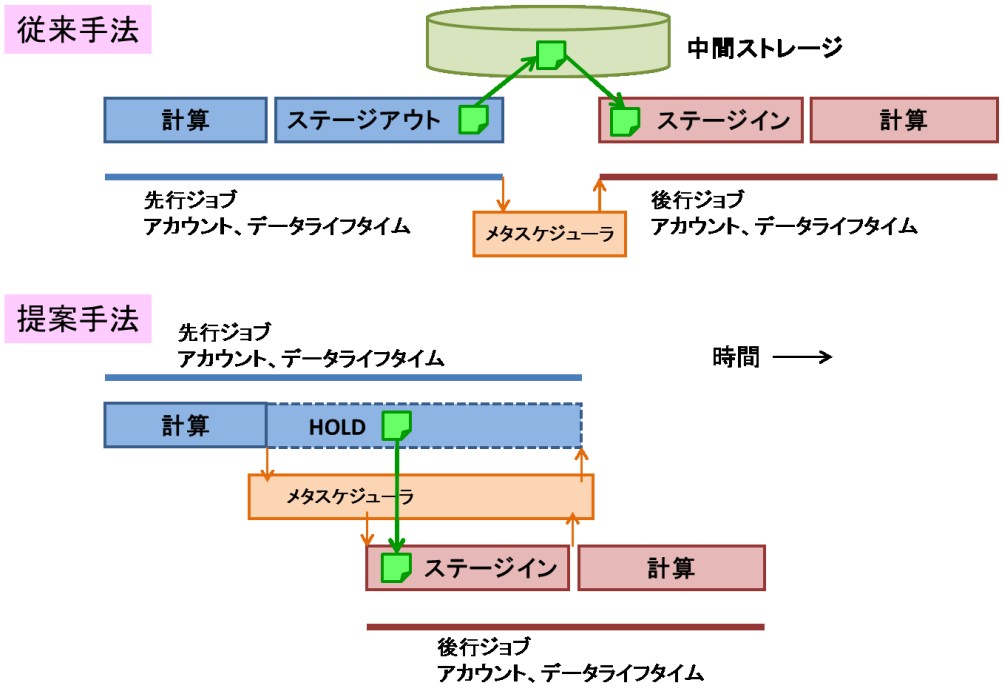


図 4-14　従来手法と提案手法のジョブの流れ

4.2.2.2. 実現方法

これら機能要件の実現方法を検討する。図 2-1 に示した HPCBP のジョブステートモデルでは、データステージングは *Running* ステートに含まれおり、外部からはデータステージングを実行しているのか、計算を実行しているのか区別できない。BES の拡張機能を利用して、*Running* ステートにデータステージイン、実行、データステージアウトのサブステートを設ければ、外部から状態を区別することはできるが、流れは制御できないので、上記のような同期機構を実現することはできない。アプリケーションに同期機構を埋め込む方法も考えられるが、アプリケーションを実行する計算ノードは組織のファイヤーウォールの内側にあり外部アクセスが許されない場合が多いので、両コンピュータ資源が別サイトにある場合を考えると、この方法は使用できない。また、外部にアクセスが許されているコンピュータ資源のジョブ実行サービス間で同期機構を持つことも考えられるが、コンピュータ資源同士の制御という従来にはない標準仕様を新たに策定しなくてはならず、また複雑な依存関係があるジョブの場合、各コンピュータ資源での制御が難しくなることなどの理由から得策ではない。そこで、従来のジョブステートモデルを細かく分割し、分割したステートを外部から制御可能とすることにより、上位のメタスケジューラが依存関係に従った同期制御することを提案する。図 4-15 に提案のジョブステートモデルを示す。具体的には、BES の *Running* ステートを *DataStageIn*、*Computing*、*DataStageOut* の 3 つに分割する。さらに従来 *Finished*、*Cancelled*、*Failed* ステートの動作に含まれていた、ジョブの一時作業領域を消去する動作と各ステートの情報を保持する動作とを分離し、一時作業領域を消去するジョブステートを *Finalizing*、各終了状態を保持するジョブステートを *Finished*、*Cancelled*、*Failed* と再定義した。*DataStageIn*、*Computing*、*DataStageOut*、*Finalizing* では、ジョブ投入時の指示によりこれらステートへ遷移したときにその実行前に HOLD させることができるものとする。また、この HOLD 状態は外部からのオペレーションで解除可能とする。さらに、状態遷移したときに上位のメタスケジューラなどに通知する機能を持つものとする。

なお、本論文の課題とは直接関係ないが、*Finished*、*Cancelled*、*Failed* ステートを一時作業領域消去動作と終了状態保持とに分離することにより、従来取得できなかった *Failure* や *Cancel* 発生時までの途中結果も取り出すことができるようになり、デバッグや障害調査にも有効である。

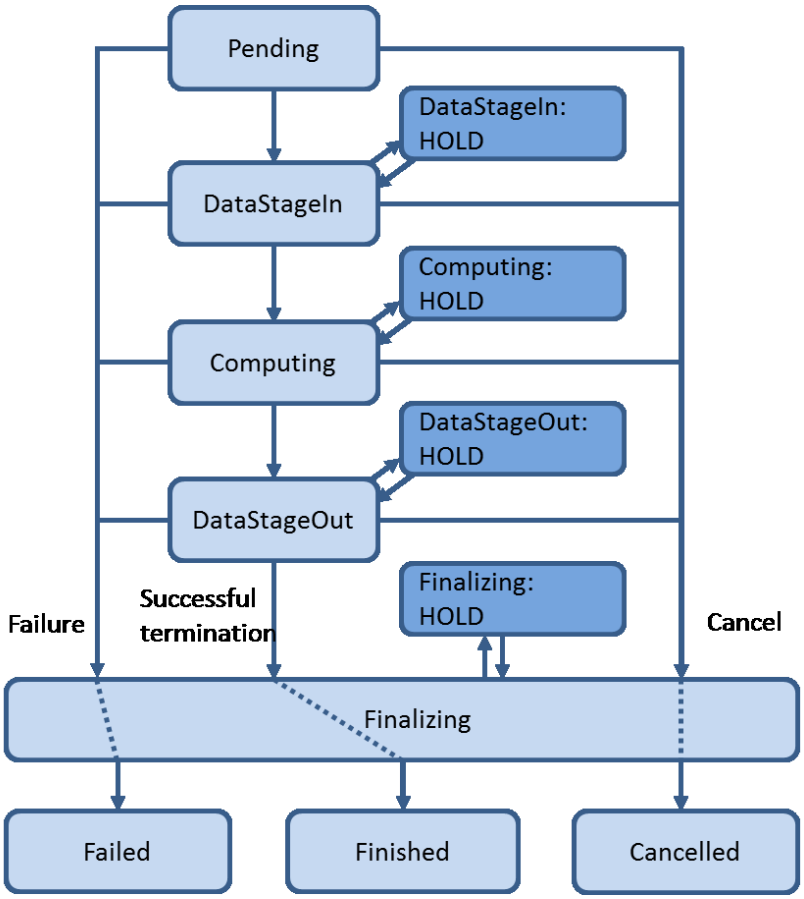


図 4-15　提案のジョブステートモデル

4.2.2.3. 基本動作

図 4-16 に依存関係のあるジョブにおける提案方法の動作例を示す。ジョブの流れは上位のメタスケジューラが制御するものとする。1) メタスケジューラはジョブ 1 をブローカリングし投入する。このとき、*Finalizing* で HOLD する指定を行う。また、計算結果の *DataStageOut* は指定しない。2) ジョブ 1 は *Finalizing* に遷移したときにジョブを HOLD しメタスケジューラに通知する。*Finalizing* に遷移すれば、*Computing* のエラーや警告はすべて出つくしているので、その終了状態も含めた通知を行う。ジョブ 1 の *Finalizing* はまだ実行されていないため、コンピュータ資源上の一時アカウントも作業領域にある計算結果も残っている。3) メタスケジューラはジョブ 1 の HOLD 通知を受け、その終了状態が正常であれば、ジョブ 2 をブローカリングし投入する。このとき、*Computing* に遷移したら通知するよう指定する (HOLD は指定しない)。4) ジョブ 2 は、ジョブ 1 のコンピュータ資源から計算結果を *DataStageIn* する。5) ジョブ 2 はステートが *Computing* に遷移したら、これをメタスケジューラに通知する。この時点で *DataStageIn* のエラーや警告はす

べて出つくしているので、ジョブ 1 の計算結果はジョブ 2 のコンピュータ資源に正常に転送されたことは保障されている。ジョブ 2 はそのまま実行を継続する。6) メタスケジューラはジョブ 2 からの *Computing* への遷移通知を受け、*DataStageIn* が正常に終了していればジョブ 1 の HOLD を解除する。7) ジョブ 1 のコンピュータ資源は *Finalizing* を実行して作業領域などを消去しジョブを終了する。2) で *Computing* が正常終了しなかった通知を受けた場合はジョブを継続しないことも可能であるし、6) で *DataStageIn* が正常終了しなかった場合は、他のコンピュータ資源にジョブ 2 を投入するなども可能である。

このように、提案手法によれば一時アカウント管理を使用する場合でも、ファイルの引き渡しを必要とするジョブのコンピュータ資源間での直接転送が可能となる。また、途中でエラーが発生したときの、ジョブの流れ制御やファイルの保障も可能となっている。

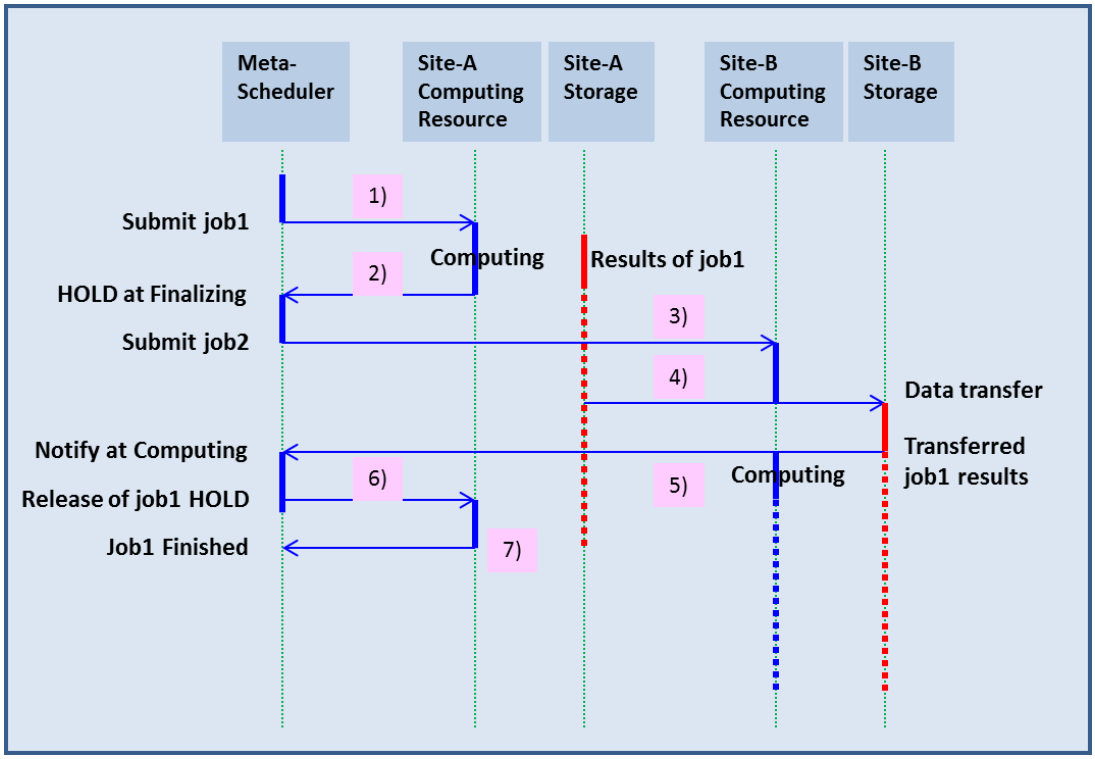


図 4-16　提案の依存関係のあるジョブのデータステージングの制御例

4.2.2.4. 静的アカウント資源と一時アカウント資源が混在するときの動作

従来手法と提案手法で、依存関係のあるジョブを一時アカウントと静的アカウントのコンピュータ資源に投入する場合を比較する。まず、従来手法を使用して、一時アカウント資源で先行ジョブを、静的アカウント資源で後行ジョブを実行する場合を考える。一時

アカウント資源の計算結果はジョブ実行中でしか有効でないため、ジョブ実行の一部として計算ステップ終了後に結果を転送するしかない。この転送先はメタスケジューラが先行ジョブ投入時に JSDL に書き込む。したがって、ジョブ資源間で直接データステージングを行うには、先行ジョブの投入時に後行ジョブの資源と転送すべき作業領域が決定されている必要がある。しかし、一般に HPC ジョブの実行時間は数日から数か月と長くなることがあるので、先行ジョブ投入時に後行ジョブの資源を決定すると、後行ジョブ実行開始時には資源が停止している場合や非常に混雑している場合などがあるため得策ではない。このため、図 4-13 に示す一時アカウント資源間でのデータステージングのように、一時ストレージを使用せざるを得ない。逆に、静的アカウント資源で先行ジョブを一時アカウント資源で後行ジョブを実行する場合、先行ジョブの計算結果はジョブ終了後も残っているので、先行ジョブ投入時に後行ジョブ資源を決定する必要はない。後行ジョブはそのコンピュータ資源のアカウントタイプとは無関係に先行ジョブのコンピュータ資源から直接データステージング可能である。この動作は図 4-12 の静的アカウント資源間のデータ転送と同様にデータ転送回数を 1 回とすることができる。このため、依存関係のあるジョブでは、常に先行ジョブは静的アカウント資源で実行するようにブローカリング機能を限定すれば問題を回避できる。しかし、資源を限定することは、資源利用の自由度が低下し利用効率を低下させる原因となるため現実的ではない。したがって、従来手法で依存関係のあるジョブを実行するには、アカウントタイプが異なる資源がブローカリングされても、常に図 4-13 のような一時ストレージを使用したデータステージングを行う。このため、背的アカウント資源のグリッド環境のユーザにとって、他グリッドとのインターオペレーションで一時アカウント資源が含まれると、データステージング回数が 1 回増加してしまう問題が発生する。

一方、提案手法では資源のアカウントタイプによらず、図 4-16 に示すシーケンスで、図 4-17 に示す動作を行うことが可能である。基本動作で説明したように、たとえ先行ジョブが一時アカウント資源であってもジョブを HOLD することによって、計算結果を保持し続けることができる。また、静的アカウント資源の場合 HOLD 状態の有無によらず計算結果は保持されている。したがって、資源のアカウントタイプに依存せずに図 4-17 の動作を実現でき、データステージング回数を 1 回とすることができる。

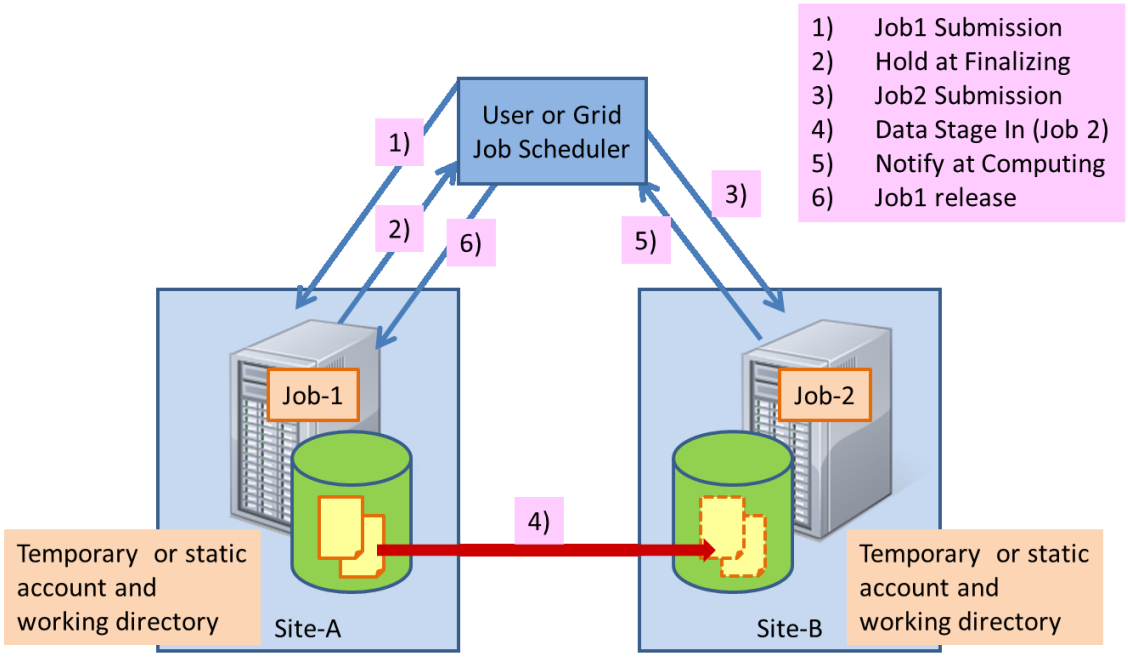


図 4-17　提案手法による資源アカウントタイプが混在した場合のデータステージング

4.2.2.5. HOLD による資源利用効率への影響

ジョブの HOLD 機能をサポートすることにより、資源の利用効率が低下することがある。これを防ぐためには、1）課題のジョブのデータ転送と第三者ジョブの実行が同時に可能な独立したジョブ実行とデータ転送の管理機能、2）データ転送機能において JSDL のソースとディスティネーションで指定された資源以外にも転送可能とする第三者転送機能、3）グリッド環境の構成要件として従来手法と同様に一時ストレージとして使用できるグリッドワイドなストレージを用意することが必要である。1）は通常のグリッドミドルウェアで多用される普通の方式であり、2）は GridFTP で実装されている機能であり, 3）はグリッド環境の構築ではグリッドワイドな共有ストレージをもつことは通常であり、無理のない提案であると考える。

これら提案手法にしたがった、先行ジョブ、後行ジョブのコンピュータ資源の計算機、ストレージの占有量、利用効率についていて議論する。これらの他に、各ジョブの管理情報を保持するデータベース資源も HOLD の影響を受けるが、現代の計算機システムが持つ大規模ストレージではジョブの管理情報は無視できる容量のため検討対象外とする。本提案手法の HOLD は、バッチスケジューラのジョブホールドと異なり、計算実行途中ではなく計算実行後の HOLD である。このため、上記提案要件 1）により計算実行とデータ転送を独立して管理、実行できれば、課題ジョブの Finalizing の HOLD でデータステージングの終了を待つ状態でも、課題ジョブと関係ない第三のジョブが実行可能である。したがっ

て、本提案の HOLD 状態での計算機資源の占有はなく利用効率の低下はない。ただしこれは、ストレージに第三のジョブが必要とする空き容量がある場合である。このため、ストレージの空き容量の状態により区分して議論する。なお、いずれの場合でも本手法の HOLD 状態は計算ステップ実行終了後であるため計算機の占有はない。ここで、ストレージ資源の空きとは、下位バッチスケジューラのキュープロパティとしてジョブに許される最大限のストレージ容量、または第三者ジョブの実行ドキュメントに記述されたストレージ容量要件の内、少ない方の容量分の空きがあることを意味する。また、従来手法との比較議論を容易にするために、従来手法における一時ストレージへのデータ転送速度と提案手法における後行ジョブ資源への直接データ転送速度が同じであると仮定する。

- **先行ジョブ、後行ジョブのコンピュータ資源ともストレージに空きがある場合**

上記のように、先行ジョブのコンピュータ資源では、計算が終了しだい計算機は解放され、さらにストレージにも空きがあるので、直ちに下位バッチスケジューラ上のキューにある第三者ジョブの実行が開始される。このため、HOLD 状態での先行ジョブの資源の計算機の占有はなく利用効率の低下はない。また、後行ジョブのコンピュータ資源でも、後行ジョブの実行に必要なストレージの空きがあるため、たとえ後行ジョブのコンピュータ資源で他のジョブが実行中でも、データステージングは先行して実行可能である（この場合は、後行ジョブのデータステージインが終了した時点での HOLD が必要)。このため、他のジョブが終了次第、計算実行を開始でき計算機の利用効率の低下はない。むしろ、後行ジョブの投入は従来方法より早いため、その分データステージインと計算を早く開始できるため利用効率が向上する。

- **先行ジョブコンピュータ資源のストレージには余裕がないが、後行ジョブコンピュータ資源のストレージに余裕がある場合**

先行ジョブの計算が終了しても、ストレージに空きがなければ第三者ジョブの実行は開始できない。先行ジョブの計算結果が転送されればストレージは解放され第三者ジョブの実行は開始される。したがって、従来方法と提案方法の利用効率の比較はデータ転送開始までの時間差で議論する。従来方法では計算終了後直ちにデータを一時ストレージにデータステージアウトできるが、提案方法では、Finalizing の HOLD のメタスケジューラへの報告、ジョブ 2 のコンピュータ資源のブローカリング、ジョブ 2 のジョブ投入、ジョブ 2 からのデータステージイン開始までの時間が必要となる。この時間はメタスケジューラの実装に依存するので一概には議論できないが、参考までに著者のプロトタイプ実装では約 40 秒であった。この遅延は、チューニングにより低減が可能である。また、ストレージ資源の空き状態が問題となるような HPC ジョブが生成する大規模データは数 GB から数 TB

となるため、この程度の遅延はデータ転送時間に比べ小さな値と考えられる。したがって、利用効率の低下は少ないと考えられる。また、計算結果サイズが小さい場合は、ユーザがあらかじめ認識できるはずであり、JSDL の HOLD エレメント指定を外して従来手法の動作をさせることにより、従来と同じ資源利用効率とすることも可能である。

- **先行ジョブコンピュータ資源のストレージには余裕はあるが、後行ジョブのコンピュータ資源のストレージに余裕がない場合**

上記のように、提案手法の資源利用効率は先行ジョブのストレージのデータを如何に早く後行ジョブの資源に転送できるかに依存する。このため、後行ジョブのストレージに空きがない場合、データステージングが開始できず利用効率が大幅に低下することが考えられる。そこで、このような場合、従来手法と同様に先行ジョブのデータを一時ストレージに転送する制御を行う。ただし、従来手法と異なり、先行ジョブがステージアウトするのではなく、後行ジョブが GridFTP などの第三者転送機能を用いて実現する。後行ジョブが自身の資源のストレージの空き状態に従い、直接自身のストレージに転送するか、一時ストレージに一旦転送するかを決定する。一時ストレージに転送した場合、自身のストレージに空きができしだい一時ストレージから転送して計算を実行する。この動作は後行ジョブのコンピュータ資源側で判断、実行することが可能なため、ジョブ記述、ジョブ投入インターフェース、メタスケジューラの動作には影響しない。後行ジョブのコンピュータ資源の利用効率については、従来手法でも後行ジョブのコンピュータ資源のストレージに空きがない場合、一時ストレージからのデータステージングは開始できないため、提案手法の資源の利用効率は従来方法と同等と考えられる。

- **両方のコンピュータ資源のストレージに余裕がない場合**

上記のように、資源の利用効率は後行ジョブの資源の空き状態に大きく依存するので、上記「先行ジョブストレージ資源に余裕はあるが、後行ジョブストレージ資源に余裕がない場合」と同じである。

以上より、結論として提案方法の資源の利用効率は従来方法より高いか、若干低い程度であり、大きく低下する場合は手動もしくは自動で従来手法の動作に切り替えて回避することができる。一般に現在のコンピュータ資源は複数の大規模ジョブを実行するのに十分なストレージを持っており、提案手法を使用すれば利用効率が向上する可能性が高い。このため、資源の利用効率の面でも提案手法は有効であると考える。

4.2.3. 実装設計

4.2.3.1. 基本設計

特定のグリッド環境でのみ提案手法サポートしても、インターオペレーション環境では課題の解決にはならない。インターオペレーション環境のすべてのグリッドミドルウェアがサポートする必要がある。そこで、本提案の要件と次世代インターオペレーション仕様として検討されている PGI 仕様の要件とを比較した。PGI の機能要件で提案手法が実現できれば、PGI 準拠のミドルウェアでは本提案が動作する可能性があるためである。結論として､PGI 要件は提案手法を実現するための要件をおおよそ満たしていることが判明した。不足している要件は PGI-WG に提案していく。本節では、この比較と PGI 要件を基にした設計仕様について説明する。

まず、提案手法を実現するために必要な機能要件を以下にまとめる。

(a) BES の Running ステートは DataStageIn、Computing、DataStageOut の 3 つに分割でき、さらに Finished ステートはクリーンナップ処理の Finalizing ステート終了状態を保持する Finished ステートに分割できること
(b) *Finished* を除くこれらステートで、ステート動作実行前に HOLD 可能であること
(c) ジョブ投入時にユーザによる HOLD ポイントの指定が可能であること
(d) 外部から HOLD 状態を解除できること
(e) ステートの遷移をメタスケジューラなどに通知可能なこと
(f) データステージングには第三者転送機能がサポートできること

PGI 要件や PGI-WG での議論は、OGF の Web ページで確認できる。提案の要件を PGI 要件が満たすかを確認するため、表 4-6 に提案の要件と関連する PGI 要件の対応を示す。要件のレベルでは、PGI 要件は提案の要件をほぼ満たしている。(a)の Finished ステートについては満たされていないが、これは PGI-WG に提案する予定である。

次に設計仕様の検討を行った。PGI 仕様を構成する BES などの個別機能の仕様は、PGI-WG の要件を基に各担当 WG が検討しているが、まだ決定に至っていない。しかし、PGI-WG では要件を検討するにあたり想定する機能仕様の議論しており、これも公開している。ジョブのステートモデルについても議論している。この議論では、BES の *Running* ステートを、*Pre-Processing*、*Processing*、*Post-Processing* ステートに分割し、これらステートと *Pending* ステートで HOLD 可能としている。また､さらに柔軟なデータステージング、エラーリカバリのために、3 階層の複雑なジョブステートモデルを議論している。*Pre-Processing*、*Processing*、*Post-Processing* は、実質的にはそれぞれ提案手法の *DataStageIn*、*Computing*、*DataStageOut* に相当すると考えられる。しかし、PGI-WG 議論のステートでは、提案手法で必要としている作業領域を削除する前で HOLD することができない。また、*Pending* ステートで HOLD 可能とする意味も不明である。そこで、実

装のジョブステートモデルは、図 4-15 の提案のジョブステートモデルを採用した。

表 4-6　提案の要件と PGI 要件の対応

提案	Requirement	Req.ID
(a),(b)	Following Activity states MUST permit 'Hold' : 'Pre-processing', other active states, 'Post-processing'	129
(b)	The service MAY offer the hold and corresponding resume functionality of Activity processing. Change state operation to resume a hold execution and to put the Activity into 'Suspend' (maybe similar to hold).	80
(c)	When the client wants to perform client initiated data-staging-in, the client MUST also specify in the Job Description via 'Hold' points that the Activity holds.	79
(c)	The Job Description document specification MUST permit the Client to request that at specified Activity states, the Execution service sets the Activity on 'Hold'	103
(d)	In order to permit the Client to perform client-directed processing of submitted Activities (for example client-directed data staging), the Execution Service SHOULD manage 'Hold' points for Activities. Fault if not supported.	76
(e)	In order to minimize polling of the Activity Status, the Execution Service MAY implement a mechanism for notification. Asynchronous notification of Activity status (for example: e-mail, SMS, etc.) - human level notification	65
(f)	Data Staging MUST support an agreed set of protocols in PGI, e.g. HTTP(S), scp, gridftp, mailto, RNS/ByteIO	143

※　（PGI 要件は OGF PGI-WG 2010-09-24_RequirementList_v7_matrix.xls より抜粋）

4.2.3.2. JSDL の HOLD、NOTIFICATION 指定拡張仕様

提案手法、PGI とも、HOLD すべきジョブステートはジョブ投入時に指定するものとしている。これを実現するために、PGI ではジョブ定義ドキュメントで HOLD 指定することを要件としている（表 4-6 の ID: 79、103）。実装設計では、この PGI 要件に従い JSDL に HOLD 指示のための拡張を行った。また、先の後行ジョブのデータステージイン終了時の報告のように通知だけの機能も効率的な動作には必要である。このため、同様に NOTIFICATION 指示のための拡張を行った。これを図 4-18 に示す。JSDL 仕様ではジョブ記述を、<Application>、<Resources>など情報カテゴリごとにサブセクションを分けている。従来の JSDL のカテゴリの中には HOLD のようなジョブの流れ制御のカテゴリはないため、サブセクション<JobFlow>を設け、その中のサブセクション<HoldPoints>で HOLD させるステートを記述し、<NotificationPoints>で通知するステートを記述する仕様とした。なお、この拡張は JSDL の拡張仕様規約にしたがい、独立したスキーマとした。

```
<JobDefinition xmlns="http://schemas.ggf.org/jsdl/2005/11/jsdl">
 <JobDescription>
  <JobIdentification>
   ...
  </JobIdentification>
  <Application>
   ...
  </Application>
  <Resources>
   ...
  </Resources>
  <DataStaging>
   ...
  </DataStaging>
  <jsdl-pgi:JobFlow:xmlns:jsdl-pgi="http://www.naregi.org/jsdl/2010/04/jsdl-pgi">
   <jsdl-pgi:HoldPoints>
    <jsdl-pgi:ActivityState>Pending</jsdl-pgi:ActivityState>
    <jsdl-pgi:ActivityState>DataStageIn</jsdl-pgi:ActivityState>
    <jsdl-pgi:ActivityState>Computing</jsdl-pgi:ActivityState>
    <jsdl-pgi:ActivityState>DataStageOut</jsdl-pgi:ActivityState>
    <jsdl-pgi:ActivityState>Finalizing</jsdl-pgi:ActivityState>
   </jsdl-pgi:HoldPoints>
  </jsdl-pgi:HoldPoints>
  <jsdl-pgi:NotificationPoints>
    <jsdl-pgi:ActivityState>Pending</jsdl-pgi:ActivityState>
    <jsdl-pgi:ActivityState>DataStageIn</jsdl-pgi:ActivityState>
    <jsdl-pgi:ActivityState>Computing</jsdl-pgi:ActivityState>
    <jsdl-pgi:ActivityState>DataStageOut</jsdl-pgi:ActivityState>
    <jsdl-pgi:ActivityState>Finalizing</jsdl-pgi:ActivityState>
   </jsdl-pgi:NotificationPoints>
  </jsdl-pgi:JobFlow>
 </JobDescription>
</JobDefinition>
```

図 4-18　HOLD、NOTIFICATION ポイントを指定する JSDL 拡張

4.2.3.3. BES 拡張オペレーション仕様

PGI では HOLD を解除するには、ジョブ実行サービスのオペレーションでジョブステート遷移を指示することで行うことを要件としている（表 4-6 の ID: 80）。提案手法でも同様にジョブ実行サービスのオペレーションで HOLD 状態を解除することを想定している。現在の BES 仕様には、ジョブステートを変化させるオペレーションは存在しないため、新たなオペレーション ChangeActivityState を追加定義した。図 4-19 にオペレーション詳細を示す。引数は、ジョブの EPR と遷移させたいジョブステートであり、現在のステートから指定されたステートへの遷移が許されない場合はエラーとなる。ジョブの HOLD を解除する場合は、HOLD しているステートの直後のステートを指定する。なお、*Cancelled* ステートへ遷移は既存の BES オペレーションである TerminateActivities オペレーションを使用する。

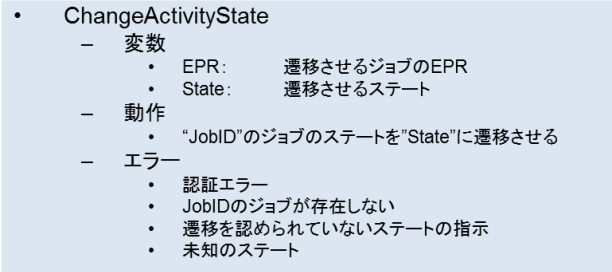
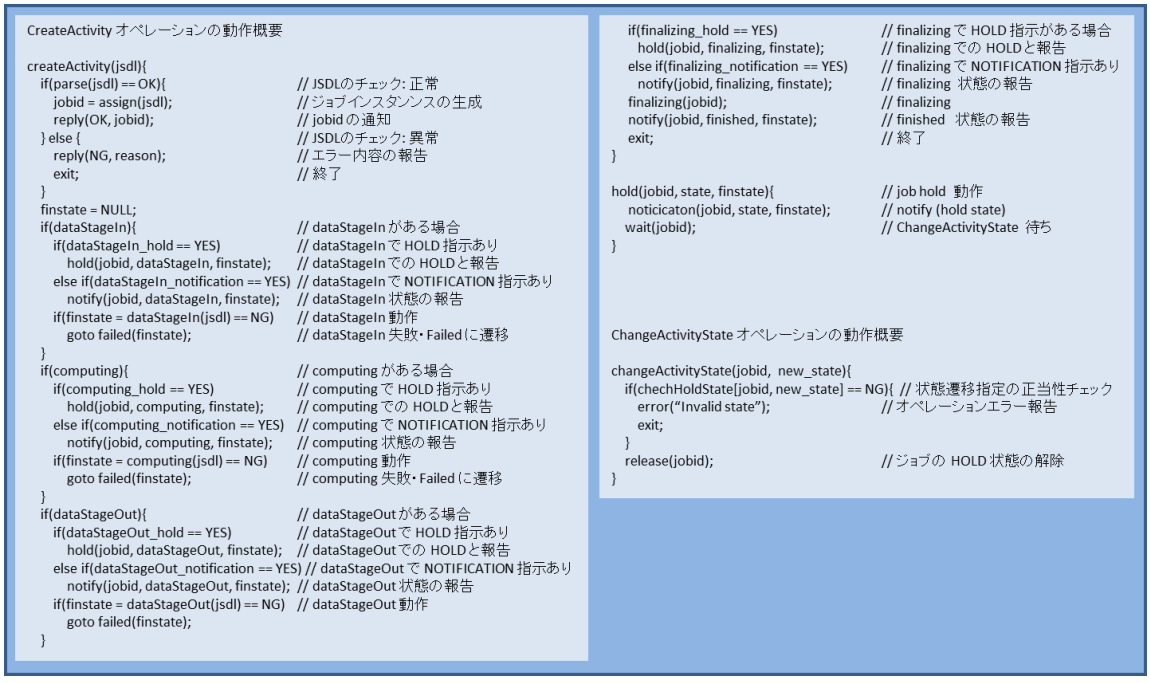


図 4-19　PGI 拡張 BES オペレーション

ChangeActivityStatus オペレーション以外のインターフェース仕様は、従来の BES オペレーションと同じである。しかし、ジョブを投入するためのオペレーションである CreateActivity では HOLD 機能と NOTIFICATION 機能をサポートするために、従来と動作が異なる。動作概要を図 4-20 に示す。JSDL に HoldPoints や NotificationPoints が指定されていると、ジョブがそのステートに遷移したときに、HoldPoints の場合はジョブステートの動作を実行する前にジョブを HOLD させメタスケジューラに報告し、NotificationPoints の場合はメタスケジューラに報告するがジョブの動作は継続する。両方が指定された場合は HoldPoints 指定が優先される。HOLD した場合は、メタスケジューラから ChangeActivityState、または TerminateActivities オペレーションが投入されるのを待つ。



CreateActivity オペレーションの動作概要

```
createActivity(jsdl){
  if(parse(jsdl) == OK){                    // JSDLのチェック: 正常
    jobid = assign(jsdl);                   //ジョブインスタンンスの生成
    reply(OK, jobid);                       // jobid の通知
  } else {                                  // JSDLのチェック: 異常
    reply(NG, reason);                      // エラー内容の報告
    exit;                                   // 終了
  }
  finstate = NULL;
  if(dataStageIn){                          // dataStageIn がある場合
    if(dataStageIn_hold == YES)             // dataStageIn で HOLD 指示あり
      hold(jobid, dataStageIn, finstate);   // dataStageIn での HOLDと報告
    else if(dataStageIn_notification == YES) // dataStageIn で NOTIFICATION 指示あり
      notify(jobid, dataStageIn, finstate); // dataStageIn 状態の報告
    if(finstate = dataStageIn(jsdl) == NG)  // dataStageIn 動作
      goto failed(finstate);                // dataStageIn 失敗・Failedに遷移
  }
  if(computing){                            // computing がある場合
    if(computing_hold == YES)               // computing で HOLD 指示あり
      hold(jobid, computing, finstate);     // computing での HOLDと報告
    else if(computing_notification == YES)  // computing で NOTIFICATION 指示あり
      notify(jobid, computing, finstate);   // computing 状態の報告
    if(finstate = computing(jsdl) == NG)    // computing 動作
      goto failed(finstate);                // computing 失敗・Failed に遷移
  }
  if(dataStageOut){                         // dataStageOutがある場合
    if(dataStageOut_hold == YES)            // dataStageOut で HOLD 指示あり
      hold(jobid, dataStageOut, finstate);  // dataStageOutでの HOLDと報告
    else if(dataStageOut_notification == YES) // dataStageOut で NOTIFICATION 指示あり
      notify(jobid, dataStageOut, finstate); // dataStageOut 状態の報告
    if(finstate = dataStageOut(jsdl) == NG) // dataStageOut 動作
      goto failed(finstate);
  }
```

```
  if(finalizing_hold == YES)                // finalizing で HOLD 指示がある場合
    hold(jobid, finalizing, finstate);      // finalizing での HOLDと報告
  else if(finalizing_notification == YES)   // finalizing で NOTIFICATION 指示あり
    notify(jobid, finalizing, finstate);    // finalizing 状態の報告
  finalizing(jobid);                        // finalizing
  notify(jobid, finished, finstate);        // finished 状態の報告
  exit;                                     // 終了
}

hold(jobid, state, finstate){               // job hold 動作
  noticicaton(jobid, state, finstate);      // notify (hold state)
  wait(jobid);                              // ChangeActivityState 待ち
}
```

ChangeActivityState オペレーションの動作概要

```
changeActivityState(jobid, new_state){
  if(chechHoldState[jobid, new_state] == NG){ // 状態遷移指定の正当性チェック
    error(“Invalid state”);                 // オペレーションエラー報告
    exit;
  }
  release(jobid);                           // ジョブの HOLD 状態の解除
}
```

図 4-20　CreateActivity、ChangeActivityState オペレーションの動作概要

4.2.3.4. メタスケジューラの動作

メタスケジューラは図 4-16 に示したワークフローを実現するよう制御する。依存関係のあるジョブに対するメタスケジューラの制御概要を図 4-21 に示す。ジョブ 1 を実行するための資源をブローカリングしジョブ 1 を投入する。ジョブ 1 が Finalizing の HOLD 状態になるまで待ち、HOLD 報告を受けたら、計算が正常に終了したことを確認する。正常に終了していたら、ジョブ 2 を実行するための資源をブローカリングしジョブ 2 を投入する。ジョブ 2 が Computing 状態に遷移した Notification を待ち、正常に遷移したことが確認できたらジョブ 1 の HOLD を解除する。ジョブ 2 の終了を待ち、ワークフロージョブとしての実行を終了する。

依存関係のあるジョブに対するグリッドスケジューラの動作概要

```
wf_submit(job1.jsdl , job2.jsdl){
  jobid1 = submit (job1.jsdl);                    // job1の資源ブローカリングと投入
  wait_nofitication(jobid1);                      // job1の HOLD待ち
  if(notification .jobid1== OK){                  // job1の計算結果
    jobid2 = submit (job2.jsdl);                  // 正常: job2の資源ブローカリングと投入
  else {
    error(notification.jobid1, NULL);             // 異常: job1異常終了報告
    exit;                                         //       ワークフロージョブ終了
  }
  wait_notification(jobid2);                      // job2  の NOTIFICATION待ち
  if(notification.jobid2== OK){                   // job2 の DataStageIn結果
    release(jobid1);                              // 正常: job1の HOLDをリリース
    wait_notification(jobid2);                    //       job2 の 終了待ち
  } else {
    release(jobid1);                              // 異常: job1の HOLDをリリース
    wait_notification(jobid1);                    //       job1 の終了待ち
    error(notification.job1, notification.jobid2);//       job2異常終了報告
    exit;                                         //       ワークフロージョブ終了
  } else {
  if(notification.job2 == OK){                    // job2の終了待ち
    scceess(notification.job1, notificaiton.job2);// 正常: 正常終了報告
    exit;                                         //       ワークフロージョブ終了
  } else {
    error(notification.job1, notificaiton.job2);  // 異常: 異常終了報告
    exit;                                         //       ワークフロージョブ終了
  }
}
```

図 4-21　依存関係のあるジョブのメタスケジューラの動作概要

4.2.4. 実験

第 3 章で研究開発した HPCBP コンピュータ資源とクライアントに、上記設計仕様を実装し、ジョブ投入実験を行った。提案通り一時アカウントを想定した環境でもコンピュー

タ資源間で直接ファイル転送できることを確認し、ジョブの最終的な結果も正常であることを確認した。また性能測定を実施した。実験環境を図 4-22 に示す。NII の東京本部に提案機能を追加した HPCBP クライアントと HPCBP コンピュータ資源を、NII の千葉分館にも提案機能を追加した HPCBP コンピュータ資源を設置した。実験に使用した装置の詳細を表 4-7 に示す。

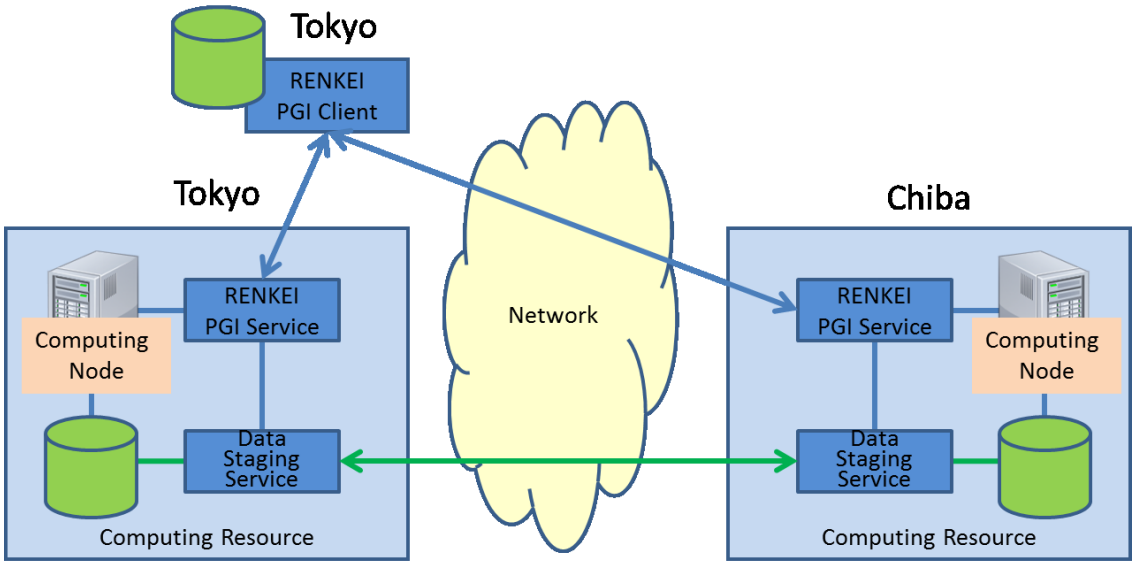


図 4-22　実験環境

表 4-7　コンピュータ資源の構成

Site	Tokyo PGI Client	Tokyo Computing Resource	Chiba Computing Resource
CPU, No. of CPU	Intel E8400 x 1	AMD 2350 x 2	Intel E5506 x 2
Clock Speed	3 GHz	2 GHz	2.13 GHz
No. of cores/node	2	8	8
No. of nodes	1	1	16
Network I/F	1Gbps	1Gbps	1Gbps
OS, Version	CentOS 5.7	CentOS 5.5	CentOS 5.4

4.2.4.1. 実験 1：動作確認とファイル転送性能測定

コンピュータ資源間の直接ファイル転送が実現されていることの確認のために、依存関係のあるジョブで実験を実施した。千葉分館のコンピュータ資源の先行ジョブでデータを

生成し、これを東京本部のコンピュータ資源に転送して、後行ジョブとしてファイルのサイズを表示するテストである。これを、従来手法と提案手法で実行しジョブ実行時間を比較した。データステージングには FTP プロトコルを使用し、従来手法で必要な一時ストレージには千葉分館に設置した RAID5 構成のファイルサーバを使用した。東京本部、千葉分館の両コンピュータ資源とも、ジョブ作業領域は各々のサイトに設置されている RAID5 構成ストレージを持つ NFS サーバ上に割り当てている。

従来手法では、”千葉コンピュータ資源”—(FTP)→”一時ストレージ(千葉ファイルサーバ)”—(FTP)→”東京コンピュータ資源”の流れであり一時ストレージを介してデータが転送され、提案手法では”千葉コンピュータ資源”—(FTP)→”東京コンピュータ資源”の流れとなるため、転送回数が少ない分だけ提案手法のジョブ実行時間は短縮されるはずである。この実験は、提案手法のプロトコルによりジョブ実行時間が短縮すること明確にすることが目的のため、実装方法により性能が大きく変わるメタスケジューラは使用せず、従来手法、提案手法のプロトコルをプログラミングしたシェルスクリプトによる簡易メタスケジューラで実験を実施した。実験環境の簡易メタスケジューラは、提案手法の HOLD 処理も含めたワークフローの流れ制御に約 0.1 秒、HPCBP インターフェースによる先行ジョブ、後行ジョブの投入にそれぞれ約 0.3 秒かかり、合計約 0.7 秒の処理時間を要する。また、コンピュータ資源の HPCBP サービスの処理時間（ジョブを受信してから実際にジョブの実行が開始されるまでの時間）は、ジョブ情報管理 DB のエントリ生成、作業領域の割当て、ユーザ認証、実行アカウントの割当て、下位バッチスケジューラのジョブスケジューリングとキューイングなどからなり、他のジョブが実行されていない状態で約 8 秒であった。この時間は本実験環境での値であり、HPCBP サービスが動作する計算機の性能、コンピュータ資源の規模、バッチスケジューラの違い、他のジョブの状態などに依存する。なお、コンピュータ資源内部だけの処理であり、外部との情報交換はないため、ネットワーク性能には影響されない。また、この時間は先行ジョブ、後行ジョブ各々に必要であるため、HPCBP サービスに使用される時間は合計で約 16 秒となる。したがって、計算、データステージング以外の処理時間は約 16.7 秒となる。実験結果を図 4-23 に示す。これらの結果は 10 回試行した平均である。千葉分館－東京本部間の FTP データ転送性能は、共有ネットワークのため 8MB/sec～20MB/sec の幅があり、千葉分館内のコンピュータ資源とファイルサーバ間もネットワークを共有しているため、10MB/sec～30MB/sec の幅がある。このように、データ転送性能に幅があること、一方のデータ転送が同一サイト内であることにより、提案手法と従来手法のジョブ実行時間の差がデータ転送一回分であることを示すことはできない。しかし、提案手法の JSDL は従来手法より DataStageing エレメントが 1 つ少なく、構造上 DataStageing エレメントの数はデータ転送回数と一致するので、この差分のほとんどはデータ転送回数の低減によるものであると考えられる。本実験により提案手法では従来性能に比べジョブの実行時間が 23%～37%向上したことが確認できた。

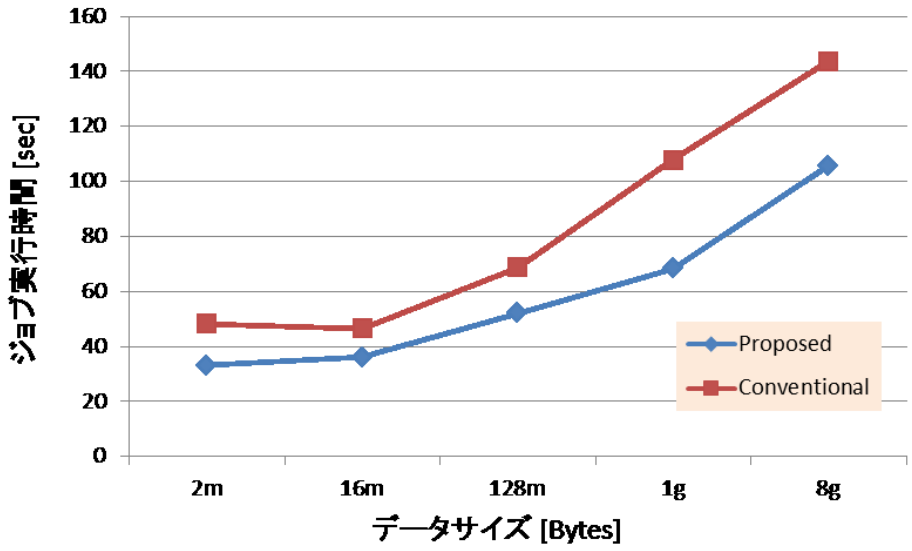


図 4-23　依存関係のあるジョブの実行時間

4.2.4.2. 実験 2：OpenFOAM

実際のアプリケーションによる実験として、千葉分館のコンピュータ資源にインストールされている OpenFOAM でダム崩壊シミュレーション 3D 拡張版の計算を実施し、東京本部のコンピュータ資源上の ParaView で可視化ムービーを作成するジョブを提案手法で実行した。ただし、ParaView はインタラクティブ操作が必要なため、千葉分館の OpenFOAM のジョブ実行開始から東京本部コンピュータ資源へのデータステージング終了までを測定した。OpenFOAM は流体シミュレーションのオープンソースソフトウェアであり、一般に広く使用されている。MPI をサポートしており、並列実行による高速化が可能である。本実験では計算規模を 50 万要素とし、計算結果は圧縮して約 350MB になる。実験環境で使用できる最大のコンピュータ資源（128 ノード）で実行したところ、計算実行だけで 1,447 秒要した。また、提案手法によるデータステージング時間は平均 30 秒程度であり、ジョブ全体の実行時間のうち、データステージングが占める割合は 2%でしかなかった。このため、提案手法の従来手法からの性能向上も最大 2%程度であり、OpenFOAM や類似の実アプリケーションでも提案手法が有効であるとは言い難い結果に見える。しかし、これは実験環境のシステムが今日の大規模コンピュータ資源に比べると小規模であり、プロセッサも古く、ネットワークも 1Gbps のイーサネットでしかないことが原因と考えられる。そこで、大規模 HPC 環境での本提案の有効性を推測するために、東京工業大学の TSUBAME 2.0 で同じ計算を実行した結果から検討した。ただし、TSUBAME 2.0 へのジョブ投入は HPCBP インターフェースではなく、TSUBAME 2.0 のジョブ投入方法に従ったものであ

る。実行したジョブの並列プロセス数は 256 であり GPU は使用していない。このときの計算実行時間は 156～400 秒の間である（混み具合により異なる）。もし、TSUBAME 2.0 の 256 コア分のシステムが著者の実験環境にあり、同じ計算性能が得られると仮定すると、ジョブ実行時間における提案手法のデータステージング時間が占める比率は 16%～7%と推測でき、また従来手法でデータステージング時間が占める比率は最大 28%～13%と推測できる。データステージングの占める比率が 28%から 16%へ低下することは有意な差であると考える。当然ながら、アプリケーションによって計算とデータ転送の比率が異なるが、比率が大きいアプリケーションでは、本提案手法によるデータ転送回数の低減が有効であることが推測できた。

4.2.5. 本節のまとめ

グリッド環境には、コンピュータ資源のユーザアカウントの管理に、従来の単体コンピュータシステムと同様の静的アカウント管理と、ジョブ毎に動的にユーザアカウントと作業領域を割当てる一時アカウント管理の 2 種類の管理方式がある。依存性のあるジョブ間でファイルを引き渡す場合、静的アカウント方式の場合はコンピュータ資源間で直接データステージングできるのに対し、一時アカウント方式の場合は中間にストレージを介した 2 回のデータステージングが必要となり、ジョブ実行時間の長期化、ネットワークトラフィックの増大を招く。これを解決するために、BES のジョブステートモデルに HOLD ステートを追加し、JSDL で HOLD させることを可能とすることにより、先行ジョブの計算が終了し、ジョブの進行に影響するエラーが出尽くした時点でジョブを HOLD させてアカウントや作業領域が無効になるのを防ぎ、この間に後行ジョブを起動することにより、コンピュータ資源間で直接データステージングできるアーキテクチャを提案した。これをインターオペレーション研究環境のミドルウェアに実装し、簡単な実験で有効性を示し、また OpenFOARM による実験を通じて実アプリケーションでも有効であることを推測できる結果を示した。

第 5 章　　おわりに

本研究では研究資源連携としてのグリッドコンピューティングの有効性を示し、これをさらに有効なものとするために異種グリッド間インターオペレーションが重要であることを示した。また、インターオペレーションの機能要件や情報表現要件を明確にし、アーキテクチャを提案した。さらに、現行のインターオペレーション仕様である HPCBP 仕様が持つ問題を明確にし、解決方法を提案して、NAREGI グリッドミドルウェアをベースにした HPCBP 準拠のミドルウェアを開発した。このミドルウェアを使用してインターオペレーション研究環境を構築し、諸外国の HPCBP 準拠のグリッドミドルウェアとのインターオペレーション実証実験に成功した。また、インターオペレーションアーキテクチャが持つコンピュータ資源のロードバランスの乱れによる性能問題を明確にし、解決方法を提案、シミュレーションによりその有効性を示した。また、ユーザアカウント管理方法の違いによりデータステージングの問題を明らかにし、解決方法を提案、プロトタイプを開発してその有効性を確認した。

第 3 章では、単一グリッド環境のジョブ投入ステップで必要となる機能、情報などの説明を行い、これをインターオペレーションに拡張するときに必要となる機能や情報を明らかにした。さらに、この必要な機能や情報の交換を満たすインターオペレーションアーキテクチャの提案を行った。また、現実には HPCBP 仕様に基づくインターオペレーションを行う必要があり、HPCBP で不足している機能や情報を明確にし、これを解決してインターオペレーションを実現する方法を提案した。また、既存のミドルウェアに HPCBP 準

拠機能を追加するときにも問題点があることを説明し、NAREGI への HPCBP 準拠機能追加を例に問題点を明らかにし、解決策を提案して実装した。この実装を使用して今後のインターオペレーション研究のための基盤環境を構築し、諸外国のグリッドとのインターオペレーション実証実験を実施し成功した。また、米国の XSEDE コンピュータ資源へのジョブ投入の性能測定を行い、米国へのジョブ投入においても実用上問題のないレベルのジョブ投入レスポンス時間であることを確認した。

第 4 章では、インターオペレーション環境で発生が予想される問題のひとつとして、ローカルグリッドと他グリッド、他グリッドと他グリッドのジョブ投入がほぼ同時に発生し、他方のジョブ投入の検出の遅れにより、コンピュータ資源間のロードバランスが乱れることが予想されることを示した。これが現実に発生する可能性があることを実験、ならびに実運用グリッドのジョブ実行ログによるシミュレーションによって示した。また、動作の詳細な検討を行い、情報サービスエージェントや情報サービスの資源情報更新間隔による遅延が原因となっていることを示した。この遅延を低減するアーキテクチャを 3 種類検討し、メタスケジューラと並行する軽量のインターオペレーション用のジョブ投入ゲートウェイを提案した。このゲートウェイは、他グリッドからのジョブを一括して受信し、コンピュータ資源にフォワードする前にメタスケジューラと共有しているジョブスケジューリングテーブルを更新する。この構造により、メタスケジューラが遅延なく他グリッドからのジョブ投入を検出することを可能とした。シミュレーションによって本提案の有効性を確認し、この問題を解決した。

また、一時アカウント管理方式をとるコンピュータ資源における、データステージング回数を低減する方法についての提案も行った。グリッド環境には、コンピュータ資源のユーザアカウントの管理に、従来の単体コンピュータシステムと同様の静的アカウント管理と、ジョブ毎に動的にユーザアカウントと作業領域を割当てる一時アカウント管理の 2 種類の管理方式がある。依存性のあるジョブ間でファイルを引き渡す場合、静的アカウント方式の場合はコンピュータ資源間で直接データステージングできるのに対し、一時アカウント方式の場合は中間にストレージを介した 2 回のデータステージングが必要となり、この 1 回多いデータステージングは、ジョブ実行時間の長期化、ネットワークトラフィックの増大を招く。この解決方法として、BES のジョブステートモデルに HOLD ステートを追加、JSDL で HOLD させることを可能とすることにより、先行ジョブの計算が終了した時点で HOLD し、さらにこの間に後行ジョブを起動することにより、コンピュータ資源間で直接データステージングできるアーキテクチャを提案した。これをインターオペレーション研究環境のミドルウェアに実装し、簡単な実験で有効性を示し、また OpenFOARM による実験を通じて、実アプリケーションでも有効であることを推測できる結果を示した。

将来構想として、ジョブ投入・実行インターオペレーションだけでなく、情報交換インターオペレーションなどを実現し、単一グリッドと比べ遜色のないレベルの機能を持つ

インターオペレーションアーキテクチャを実現すべきであると考えている。これには現在検討されている、次世代インターオペレーション仕様 PGI だけでなく、更なる機能や情報交換に関する研究が必要である。さらにその延長として、グリッド間連携だけでなく、クラウド間連携、ローカルシステム・グリッド間連携、グリッド・クラウド間連携にも適用可能なインターオペレーション環境を研究すべきと考えている。

謝辞

本博士論文の作成ならびに研究は、多くの方々のご指導、ご助言、ご協力ならびに共同研究なしには遂行することはできませんでした。皆様に心よりの感謝を申し上げます。

まず、指導教員である国立情報学研究所ならびに総合研究大学院大学の三浦謙一名誉教授ならびに合田憲人教授にはご多忙中にもかかわらず多くのご指導、ご助言をいただきました。三浦先生には入学時から先生がご退官されるまでの 2 年半の間、指導教官としてご指導、ご助言をいただいたのみならず、ご退官後名誉教授となられてからも様々なご指導、ご助言をいただきました。合田先生にはご多忙中にもかかわらず、三浦先生退官後の指導教員を引き受けていただきました。先生のゼミでの本研究課題の議論、貴重なご指導、ご助言がなければ論文作成ならびに研究を遂行することができませんでした。両先生に心よりの感謝を申し上げます。

中間発表ならびに博士論文発表会においてご意見をいただき、さらにご多忙中にもかかわらず本論文の審査をしてくださいました、東京工業大学の松岡聡教授、国立情報学研究所ならびに総合研究大学院大学の漆谷重雄教授、鯉渕道紘准教授に心よりのお感謝を申し上げます。先生方の貴重なご指導、ご意見により研究を向上させ、本論文を完成させることができました。

また、本研究は国立情報学研究所が中心となって推進し、著者がメンバーとして参加させていただいた「最先端・高性能汎用スーパーコンピュータの開発利用プロジェクト NAREGI プログラム」（2003 年度～2007 年度）ならびに「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 REsources liNKage for E-scIence (RENKEI)」（2008 年度～2011 年度）の研究の一部として実施されました。両プロジェクトへの参加なしには、本研究に携ることがないばかりでなく、総合研究大学院大学へ入学し学位取得に挑むこともなかったと思います。本プロジェクトに参加する機会を与えてくださった両プロジェクトのリーダであられた三浦先生に感謝いたします。また、プロジェクトのサブリーダであられた、松岡先生、合田先生には多くのご指導、ご助言をいただきました。松岡先生には、グリッドコンピューティングに関して多くのご指導をいただき、さらに国際的なコミュニティへの参加機会をいただきました。この機会なしには、国際的なインターオペレーション研究を遂行することはできませんでした。合田先生には、さらなる国際的コミュニティの参加機会をいただき、また研究遂行のために必要な環境の取得にご尽力いただきました。この資源の環境構築と運用を通じて、本研究のみならずクラウドコンピューティングに対する知識を得ることができました。両プロジェクトは、国立情報学研究所、東京工業大学、大阪大学、宇都宮大学、筑波大学、九州大学、九州工業大学、玉川大学、産業技術研究所、分子化学研究所、高エネルギー加速器研究機構、富士通株式会社、株式会社日立製作所、日本電気株式会社の多くの先生方、研究者、開発者による共同研究の成果であり、これら多くの方々の成果なしには本研究を遂行することができませんでした。関係各位に感謝を申し上げます。特に、国立情報学研究所リサーチグリッド研究開発センター在籍時代に、ご協力、ご助言いただいた宇佐見仁英教授（現、玉川大学）、田中義一氏（現、明星大学）、鶴澤武士氏（現、日立製作所）、山田清志氏（現、日立製作所）、坂根栄作准教授、本山一隆氏には、心より感謝申し上げます。また、RENKEI プロジェクトでインターオペレーション API を研究し、さらに同時期に総合研究大学院大学の高エネルギー加速器科学研究科で学んでいた高エネルギー加速器研究機構の河井裕氏（現、日本 IBM）とは、インターオペレーションについて多くの有意義な議論を持てたことを感謝いたします。また、事務的な面で多くのサポートをいただいた、田中恵庫氏、蕪木恵美子氏、古野博子氏に心よりの感謝の意を表します。皆さんのご努力がなければ両プロジェクトのスムーズな進行はなく、本研究の推進にも多大な影響があったと思います。両プロジェクトでは、あまりのも多くの方にお世話になり、失礼ながらここにお名前を書ききれません。関連各位には、心よりお礼申し上げます。

また、本研究の国際的なインターオペレーション実験の実現には、ドイツ Jülich 研究所の Morris Ridel 氏、英国 Oxford 大学 e-Research センターの Stephen Crouch 氏、米国 Virginia 大学の Andrew Grimshaw 教授、Karolina Sarnowska-Upton 氏、その他実験に参加した各国のグリッドプロジェクトの担当者には、資源の使用や設定など多大なご協力をいただきました。皆様のご協力なしには本研究の実証実験は不可能でした。心より感謝申し上げます。

また合田先生のゼミにて、本研究について貴重なご助言をいただいた、孫顕氏、藤原一

毅氏をはじめとする合田ゼミメンバーに感謝の意を表します。

最後に、私を学生として受け入れ、勉学と研究の場を提供してくださった、総合研究大学院大学に感謝いたします。

皆様、ありがとうございました。

2012 年 9 月

佐賀　一繁

発表

- **ジャーナル論文（査読あり）**

  佐賀　一繁、合田　憲人、三浦　謙一: 「グリッドコンピューティング環境におけるジョブ間データ転送の削減」: 情報処理学会論文誌 ACS39: 2012.

- **ジャーナル論文（査読なし）**

  Satoshi Matsuoka, Kazushige Saga, Mutsumi Aoyagi: Coupled-Simulation e-Science Support in the NAREGI Grid: IEEE Computer Volume 41 Issue 11, Nov. 2008

- **国際学会発表（査読あり）**

  Kazushige Saga, Kento Aida, Kenichi Miura: “Mutual job submission architecture that considered workload balance among computing resources in the grid interoperation”: 12th IEEE/ACM International Conference on Grid Computing, Lyon, Sep. 2011.

- **口頭発表（査読なし）**

Kazushige Saga: "Kazushige Saga: “Cyber Science Infrastructure and NAREGI Grid middleware”: OGF24: Singapore: Sep. 2008.

Kazushige Saga: “Why do we need PGI?: NAREGI perspective”: OGF25: Catania: Mar. 2009.

Kazushige Saga: “Interoperation Across Production Grids”: APAN28: Kuala Lumpur: Jul. 2009.

Kazushige Saga: “RENKEI”: Para 2010 State of the Art in Scientific and Parallel Computing in Reykjavík: Jun, 2010.（招待）

Kazushige Saga: “RENKEI Status Updates”: OGF31: Taipei: Mar. 2011

- **標準化活動**

2011 年 3 月～2012 年 3 月: Open Grid Forum, Grid Interoperation Now Community Group 共同議長

参考文献

[1] I. Foster and C. Kesselman, The Grid 2, Morgan Kaufmann, 2003.

[2] I. Foster, "What is the Grid? A Three Point Checklist," gridtoday, 2002.

[3] U. Berkeley, "SETI@home," [Online]. Available: http://setiathome.berkeley.edu/. [Accessed 26 6 2012].

[4] "TeraGrid," TeraGrid, [Online]. Available: https://www.teragrid.org/. [Accessed 19 7 2011].

[5] OSG, "The Open Science Grid," [Online]. Available: https://www.opensciencegrid.org/. [Accessed 26 6 2012].

[6] "FutureGrid," FutureGrid, [Online]. Available: https://portal.futuregrid.org/. [Accessed 19 7 2011].

[7] "The Enabling Grids for E-sciencE," Enabling Grids for E-sciencE, [Online]. Available: http://www.eu-egee.org/.

[8] "Distributed European Infrastructure for Supercomputing Applications," Distributed European Infrastructure for Supercomputing Applications, [Online]. Available: http://www.deisa.eu/. [Accessed 15 1 2012].

[9] "Extreme Science And Engineering Discovery Environment," Extreme Science And Engineering Discovery Environment, [Online]. Available:

https://www.xsede.org/. [Accessed 15 1 2012].

[10] "NORDUGRID," NorduGrid, [Online]. Available: http://www.nordugrid.org/. [Accessed 15 1 2012].

[11] d-grid, "d-grid," [Online]. Available: http://www.d-grid.de/. [Accessed 26 6 2012].

[12] "European Grid Infrastructure," European Grid Infrastructure, [Online]. Available: http://www.egi.eu/. [Accessed 19 7 2011].

[13] S. Newhouse, "EGI: Federating Virtualised Resources," [Online]. Available: http://www.ogf.org/SAUCG/materials/2342/OGF-32-CloudWS-Newhouse.pdf. [Accessed 26 6 2012].

[14] "Open Grid Forum," Open Grid Forum, [Online]. Available: http://www.ogf.org/. [Accessed 15 1 2012].

[15] B. Dillaway, M. Humphrey, C. Smith, M. Theimer and G. Wasson, "HPC Basic Profile," Open Grid Forum, 2007. [Online]. Available: http://www.ogf.org/documents/GFD.114.pdf.

[16] "Production Grid Infrastructure," Open Grid Forum, [Online]. Available: http://www.ogf.org/gf/group_info/view.php¥?group=pgi-wg. [Accessed 19 7 2011].

[17] "National Research Grid Initiative," 国立情報学研究所, [Online]. Available: http://www.naregi.org/project/index.html.

[18] C. Catlett and S. Matsuoka, "Multi-Grid Interoperation Planning Meeting," 2005.

[19] "Lightweight Middleware for Grid Computing," CERN, [Online]. Available: http://glite.web.cern.ch/glite/.

[20] "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," 国立情報学研究所, [Online]. Available: http://www.e-sciren.org/index.html.

[21] "Uniform Interface to Computing ResourcesE," Uniform Interface to Computing ResourcesE, [Online]. Available: http://www.unicore.eu/.

[22] "Advanced Resource Connector," NorduGrid, [Online]. Available: http://www.nordugrid.org/arc.

[23] "HPC Server Basic Profile Web サービス運用ガイド," Microsoft, [Online]. Available: http://technet.microsoft.com/ja-jp/library/cc972837(v=ws.10).aspx. [Accessed 27 5 2012].

[24] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher and A. Savva, "Job Submission Description Language (JSDL) Specification," Open Grid Forum, 2008. [Online]. Available:

http://www.ogf.org/documents/GFD.136.pdf.

[25] M. Humphrey, C. Smith, M. Theimer and G. Wasson, "JSDL HPC Profile Application Extension," Open Grid Forum, 2007. [Online]. Available: http://www.ogf.org/documents/GFD.111.pdf.

[26] I. Foster, A. Grimshaw, P. Lane, W. Lee, M. Morgan, S. Newhouse, S. Pickles, D. Pulsipher, C. Smith and M. Theimer, "OGSA Basic Execution Service," Open Grid Forum, 2008. [Online]. Available: http://www.ogf.org/documents/GFD.108.pdf.

[27] R. Housley, W. Polk, W. Ford and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," 2002.

[28] OASIS, "Web Services Security UsernameToken Profile 1.0," OASIS, 2004.

[29] "PGI-WG GridForge," Open Grid Forum, [Online]. Available: https://forge.ogf.org/sf/projects/pgi-wg/.

[30] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, A. Merzky, J. Shalf and C. Smith, "A Simple API for Grid Applications (SAGA)," 2008. [Online]. Available: http://www.ogf.org/documents/GFD.90.pdf.

[31] "European Middleware Initiative," European Middleware Initiative, [Online]. Available: http://www.eu-emi.eu/.

[32] S. Tuecke, V. Welch, D. Engert, L. Pearlman and M. Thompson, "Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile," [Online]. Available: http://www.ietf.org/rfc/rfc3820.txt.

[33] "Globus Alliance," Globus Alliance, [Online]. Available: http://www.globus.org/.

[34] XSEDE, "XSEDE Architecture Overview & Context," [Online]. Available: http://www.ogf.org/SAUCG/materials/2342/XSEDE-context-v1.0.pdf.

[35] "Partnership for Advanced Computing in Europe," Partnership for Advanced Computing in Europe, [Online]. Available: http://www.prace-project.eu/. [Accessed 19 7 2011].

[36] "EMI Execution Service," CERN, [Online]. Available: https://twiki.cern.ch/twiki/bin/view/EMI/EmiExecution Service/. [Accessed 19 7 2011].

[37] "EMI Standard Adoptions," EMI, [Online]. Available: http://www.ogf.org/OGF32/materials/2333/2011-07-15_OGF32_EMI_Session_Standardization_Riedel_v1.pdf. [Accessed 19 7 2011].

[38] 中田 秀基, 佐藤 仁, 佐賀 一繁, 畑中 正行, 佐伯 裕治 , 松岡 聡, “NAREGI ミド

ルウェアβ-gLite 間における相互ジョブ起動実験,” 情報処理学会研究報告. 計算機アーキテクチャ研究会報告 2007(17) , 2007.

[39] "EU-IndiaGrid," [Online]. Available: http://www.euindiagrid.eu/. [Accessed 26 6 2012].

[40] S. Chattopadhyay, "Interoperability challenges in Garuda GTS and EGEE grids," [Online]. Available: http://www.euindiagrid.eu/index.php/pubblications-doc/doc_details/265-garuda-egee-interoperability-challenges-s-chattopadyay--garuda-project-leader.

[41] "Amazon Elastic Compute Cloud (Amazon EC2)," Amazon, [Online]. Available: http://aws.amazon.com/ec2/.

[42] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke and M. Xu, "Web Services Agreement Specification (WS-Agreement)," [Online]. Available: http://www.ogf.org/documents/GFD.107.pdf. [Accessed 27 6 2012].

[43] OASIS, "OASIS Web Services Business Process Execution Language (WSBPEL) TC," [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.

[44] A. Savva, "JSDL SPMD Application Extension, Version 1.0," [Online]. Available: http://www.gridforum.org/documents/GFD.115.pdf. [Accessed 27 6 2012].

[45] T. G. S. Team, "Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective," [Online]. Available: http://www-unix.globus.org/toolkit/docs/4.0/security/GT4-GSI-Overview.pdf. [Accessed 28 6 2012].

[46] S. Andreozzi, S. Burke, F. Ehm, L. Field, G. Galang, B. Konya, M. Litmaath, P. Millar and J. Navarro, "GLUE Specification," 2009. [Online]. Available: http://www.ogf.org/documents/GFD.147.pdf.

[47] R. Mach, R. Lepro-Metz and S. Jackson, "Usage Record - Format Recommendation," Open Grid Forum, 2007. [Online]. Available: http://www.ogf.org/documents/GFD.98.pdf.

[48] "BES++," Platform, [Online]. Available: http://sourceforge.net/projects/bespp/. [Accessed 19 7 2011].

[49] A. R. Alvarez, C. Smith and M. Humphrey, "BES++: HPC Profile Open Source C Implementation," [Online]. Available: http://www.cs.virginia.edu/~humphrey/papers/bespp.pdf.

[50] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell and J. V. Reich, "The Open Grid Services Architecture, Version 1.5," [Online]. Available: http://www.gridforum.org/documents/GFD.80.pdf.

[51] "Distributed Management Task Forth," Distributed Management Task Forth, [Online]. Available: http://www.dmtf.org/. [Accessed 19 7 2011].

[52] "Common Information Model (CIM)," DMTF, [Online]. Available: http://www.dmtf.org/standards/cim. [Accessed 19 7 2011].

[53] M. Morgan, A. Grimshaw and O. Tatebe, "RNS Specification," Open Grid Forum, 2010. [Online]. Available: http://www.ogf.org/documents/GFD.171.pdf.

[54] TUDelft, "The Grid Workloads Archive," [Online]. Available: http://gwa.ewi.tudelft.nl/pmwiki/.