ハイパフォーマンス
コンピューティング　87－32
（２００１．　７．２７）

Virtual Private Grid（VPG）：遠隔計算機を効率的に利用するシェル

金田憲二[†]　田浦健次朗[†]　米澤明憲[†]

複数管理ドメインにまたがる多数の計算機を安全かつ簡便に利用可能にするための Grid 計算用のシェルを設計・実装する．複数サブネットや地理的に離れた場所にまたがる多数の計算機を利用する機会が増えている．それらの計算機は複数の管理者によって管理され，安全性や管理費用の問題から，さまざまな制限が課せられている．それらを迂回する方法は管理方針毎にまちまちで，利用の手間を大幅に増大させ，結果として計算資源の利用率を著しく損ねている．本研究では、それらの迂回を自動的に行い，利用者が複数管理ドメインにまたがる多数の計算機を簡便に利用できるためのシェルを設計，作成する．

Virtual Private Grid (VPG): A Command Shell for Utilizing Remote Machines Efficiently

Kenji Kaneda,[†] Kenjiro Taura[†] and Akinori Yonezawa[†]

We design and implement a shell that can easily and securely utilize a large number of machines spread across multiple administrative domains. Today many people use a large number of machines across multiple subnets or geographically distributed places. These machines are managed by different administrators, and for security and administration cost, they impose various restrictions. A method to work around these restrictions are found on a case-by-case basis and requires human intervention. Therefore, it increases the user's cost to utilize remote machines significantly, and consequently decreases the utilization of computational resources considerably. In this research, we design and develop Virtual Private Grid (VPG), a shell that automatically works around these restrictions and can easily utilize a large number of machines in multiple administrative domains.

1. 導　　入

今日ではネットワーク技術の発達は目覚ましく，スーパーコンピュータ，クラスタ，ワークステーション，PC などが高速のネットワークで結合されるようになってきた．それにともなって利用者が多数の計算機に対してアカウントを持ち，ヘテロな環境における数百にも及ぶ地理的に分散した多数の計算機を利用する機会というものが増えてきている．

しかしそのような計算機は複数サブネットにまたがり，異なる管理者により管理されている．そして、安全性や管理費用の問題から，ホストの名前付け，通信，ログインなどに関してさまざまな制限が課せられている．制限には，DNS 名を持たないホスト，固定 IP アドレスを持たないホスト，プライベート IP アドレス，IP フィルタリング，サブネット外からのログイン禁止などがある．それらを迂回する方法は管理方針毎にまちまちで，利用の手間を大幅に増大させ，結果として計算資源の利用率を著しく損ねている．

例えば**図 1** のようなネットワークを考えてみる．3 つサブネットが存在し，そのサブネット内にはプライベート IP しか持たないホストなども存在する．この環境で任意のホスト間のジョブ投入の実現を考える．既存の方法でこれを実現するのは困難である．例を挙げると以下のような問題がある．

- 名無しのホストを指定するのが困難である．
- 場合によっては．Secure Shell（SSH）[4] などのジョブ投入ツールを多段に使う必要があり，オーバヘッドがかかる．
- 複数ホストにまたがる相互関与したジョブ投入などが行えない．

これらの問題を解決するひとつの方法としては，ホスト間でコネクションを永続的に維持し，それを利用してジョブ投入を実現する仕組みを構築することが挙げられる．しかし，任意のホスト間でコネクションを生成する

† 東京大学
University of Tokyo

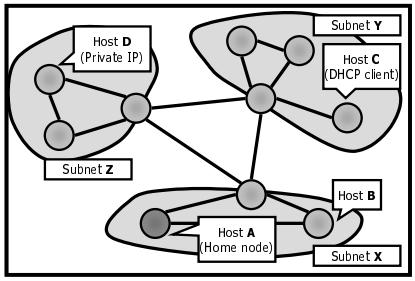

図 1 複数サブネットにまたがるネットワークの例
Fig. 1 example of network

というナイーブな方法では，計算機の台数が増えるにつれてコネクションの数が莫大なものとなってしまう．必要なコネクションのみを検出する賢いアルゴリズムが求められる．

そこで，本論文では，複数管理ドメインにまたがる多数の計算機を安全かつ簡便に利用可能にするための Grid 計算用のシェル Virtual Private Grid（VPG）を設計・実装する．最終的には複数サブネットにまたがる多数の計算機を一つのローカル計算機のように扱うことを目指す．

2 節で関連研究について説明する． 3 節で関連研究との比較を行いながら Virtual Private Grid の概要について述べる． 4 節では実装について述べ， 5 節ではその実装で用いられるアルゴリズムについて述べる． 6 節では実験内容について説明し，評価・考察を行う．最後に 7 節でまとめと今後の課題を説明する．

2. 関 連 研 究

近年では， Grid[1] という広域ネットワーク上に分散した計算，情報資源を利用して，大規模計算を実現する基盤技術が提唱されている．そういった技術の一つとして Globus[2] が挙げられる． Globus はその一部として遠隔ジョブ投入の機能を持つが，複数サブネットにまたがる資源の利用などは可能ではない．またインストールなどの際は管理方針の変更を必要とする．

Resource Manager beyond the Firewall（RMF）[3] は， Globus の Resource Manager を改良しファイアーウォール内部の資源を利用可能にしたものである．

SSH は，遠隔計算機への安全なアクセスを提供するものである．しかし数百台にも及ぶ計算機全ての有効利用を SSH で計ろうとするのは困難である．なぜなら，多数の計算機一つ一つに対してウィンドウを生成，有効利用するのは多大な利用者の手間を必要とする．かといって，ジョブごとに毎回ホストに対してコネクションを生成しては，認証などにかかる時間が多大なものとなってしまうからである．また，前節で述べたような問題にも対処できていない．

Virtual Private Network（VPN）[5] はインターネット上にプライベートなネットワークを構築することを指す．これも管理方針の変更を必要とする．

まとめると，多くのものは管理方針を変更すること無く，図 1 の様なネットワークを簡潔かつ効率的に扱う機能を提供していない．

```
path@nickname
path@nickname > file@nickname
path@nickname < file@nickname
path@nickname | path@nickname
```

図 2 シェルのシンタックス
Fig. 2 shell syntax

3. Virtual Private Grid（VPG）

3.1 概　　要

そこで，上述した問題に対処することを本研究では目指し， Virtual Private Grid を設計・実装する．具体的に VPG は以下の機能を持つ．

- DNS 名や固定 IP アドレスによらない、利用者ごとに普遍なホストの名前付け (nicknaming)
- nicknaming を用いた任意の遠隔計算機へのジョブ投入
- nicknaming を用いた任意のホスト上のファイルのリダイレクション
- nicknaming を用いた任意のホスト間のパイプによる通信機構

これらの機能は管理者によって定められた管理方針を変更することなくユーザレベルで実現可能であり，標準入力・出力を用いる既存のプログラムをネットワークを介して結合することができる．また， nicknaming を用いることによって， DNS ホスト名を持たないホストなどに対してもジョブ投入が可能となっている．最低限必要なコネクションのみを永続的に維持し，ジョブごとのコネクション生成を行わないようになっている．利用者の扱うホストも動的に追加・削除可能となっている．

3.2 シェルのシンタックス

シェルのシンタックスは**図 2** のようになっており，基本的には通常のシェルと同様である。

コマンド名のあとに `@` とホストのニックネームを加えることによって，遠隔ジョブ投入が行われる．ただし、プログラム内部での遠隔ファイルの参照や、コマンドの引数でのホストの指定は可能ではない．例えば

`gcc fileA@hostX -o fileB@hostY` というようなことは可能ではない.

3.3 ジョブ投入例

図 1 のネットワークで 2 つのジョブ投入の例を挙げながら、どのように VPG が動作するかについて説明する．利用者は HostA にログインしていると仮定する．

- `ls@HostB` は，HostA と HostB が同じサブネットに属するため，直接 HostB にジョブ投入される．通常の遠隔ジョブ投入と同様の動作となる．
- `ls@HostC | sort@HostD` では，HostA，HostC，HostD は異なるサブネットに属している．すると，外部に公開されているホストを自動的に中継して HostC にジョブは投入される． HostD に対しても同様であり，自動的に経路が検出される．また，HostC， HostD は利用者が独自に定義したニックネームによって指定することが可能である．もちろん通常のパイプ同様，並列に 2 つのジョブは実行される．

4. 実　　装

4.1 実装の概要

VPG が実際にどのようにしてコネクションの生成. ジョブ投入を実現するかについて説明する（**図 3**）.

(1) 利用者が任意の順番で，自分のアカウントの持っているホストで VPG デーモンを起動させる．その際に，そのホストのニックネームとそこから接続可能なホストのリストをデーモンに渡す．リストは利用者自身が設定・記述する必要がある．

(2) デーモンが次々とホストごとに立ち上がっていく．デーモンは隣接するホストから情報を取得し，自動的に必要なコネクションのみを作成，維持する．情報の取得，交換は，一時的なコネクションを一定時間ごとに生成することによって行う．

(3) 最終的にデーモンが全てのホストで立ち上がると，任意のホスト間の通信に最低限必要なコネクションのみが維持されることとなる．これ以降のホスト間の通信は, このコネクションを通じて行われるようになる.

(4) ユーザのログインしているホスト（ホームノード）でシェルが立ち上がる．シェルは全てのホストからコネクションに関する情報を収集し，ネットワークの位相を把握する.

(5) ジョブ投入の際などは，把握したネットワークの位相を元に，目的のホストへの最短路が計算される.

(6) デーモンの追加・削除を行うことによって，利用するホストの追加・削除が動的に行える．デーモンはネットワークの位相の変化に応じて，新たにコネクションの生成・削除を行う.

4.2 SSH の利用

ファイアーウォールの IP フィルタリングをまたいだコネクションの生成を可能にするため， SSH を用いている．空のパスフレーズによる公開鍵認証を用いることによって，ユーザによるパスワード，パスフレーズの入力を必要とせずに，認証を行う．これはセキュリティホールになる可能性もあり，代替手段をとることも考えられる.

5. アルゴリズム

ネットワークをモデル化し，生成するコネクションの選択などのために実装で用いられるアルゴリズムを記述する.

5.1 モ デ ル

ネットワークをグラフにモデル化する.

定義 5.1　ネットワークを有向グラフ $G = (V, E)$ と定義する．ただし,

$V = \{u | u$はデーモンの立ち上がるホスト$\}$

$E = \{(u, v) | u$はvへコネクション生成可能$\}$

となっている．例えば DHCP クライアント u は

$\forall v \in V, (v, u) \notin E$

という条件によって表すことができる．プライベート IP しかもたないホスト u は

$\forall v \in$ (プライベートネットワーク外のホストの集合),

$(v, u) \notin E$

と表すことができる.

また，以下のような性質を持つ.

- 各々のノードが普遍かつ固有の値を持つ（ニックネームと対応）.
- 各々のノードが局所状態をもち，その集合によって大域状態が決定される．隣接するノードの局所記憶の読み込み，自分の局所記憶への書き込みを行うことによってのみ，通信を行うことができる．また現在の局所状態から次の局所状態へと有限時間内に遷移するとする.
- ネットワークの位相が動的に変化し，ノード，コネクションの増減がある．ただしその変化は有限で，最終的には停止するとする.

5.2 Self-Stabilizing Spanning Tree Algorithm

任意のホスト間を通信可能にするために最低限必要な

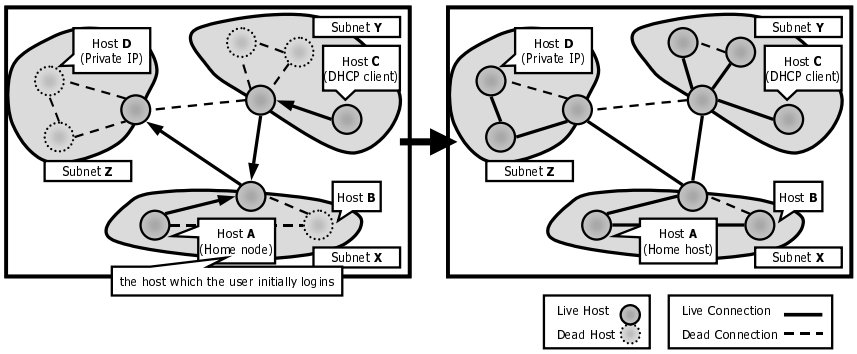

図 3 動作の流れ
Fig. 3 overview of implementation

コネクションの検出を行うため，分散・非同期な環境において極大木を構築するアルゴリズム[6)7)] を用いる．これを用いて，各々のホストが自律してコネクションの検出を行う．そして，デーモンの立ち上がる順序に依存せずネットワークの構築が可能で，かつホストの増減に動的に対応可能となる．

5.2.1 アルゴリズムの概略

任意の状態から有限回の遷移で適切な状態（極大木が構成されている状態）に到達するというアルゴリズムであり，各々のノードが自律的に動作し，木を構成しようとする．そして森が最終的に一つの木に結合することにより，極大木が構築される．

各々のノードが優先度という値を持つ．構成される木の中で最も高い優先度を持つノードがルートとなり，ルートの優先度がその木に属する全てのノードの優先度となる．2 つの木が結合後は，より高い優先度を持つノードがその木のルートとなる．最終的には最も高い優先度を持つノードをルートとする木が構築される．これを利用して，優先度の更新の不可能な状態であることを，極大木の構築された状態であるための必用条件としている．位相の動的な変化に対応するため，必要に応じて異常事態を検出し，局所的に状態の初期化を行う．その際に，Ghost Root 問題[6)7)] にも対応可能にする．Ghost Root 問題とは，状態の初期化などが正しく動作せず，ルートに関して誤った情報が永続的に保持されてしまい，極大木が構成されない問題のことを指す．

5.2.2 アルゴリズムの詳細

各々のノードは**図 4** のような変数を持つ．ID，$Root$，$Parent$，$Distance$ といった変数は木を構築するために用いられ，実際に生成される木は $Parent$ の集合によって表される．ID はタップルとなっていて，あとで述べるように動的に値の追加が行われる．ID と $OtherTrees$ によって GhostRoot 問題への対処を行っている．

UID_u：ノードの持つ固有かつ普遍な値
$Nbrs_u := \{v|(u,v) \in E \vee (v,u) \in E\}$

$Root_u$：ノード u の属する木のルートの UID
$Parent_u$：u の親．この集合によって木を表す
$Distance_u$：ルートからノード u までの hop 数
ID_u：タップル．優先度に影響

$OtherTrees_u$：同じ ID を持つ木の検出

図 4 ノード u のもつ局所変数
Fig. 4 Local variables of node u

まず，ノードの優先度について定義する．

定義 5.2 ID の順序関係
$ID_u = (x_1, \ldots, x_i)$，$ID_v = (y_1, \ldots, y_j)$ とすると，

$$\begin{aligned} ID_u \prec_w ID_v &\equiv (i < j) \wedge \\ &\quad ((x_1, \ldots, x_i) = (y_1, \ldots, y_i)) \\ 2ID_u \prec_s ID_v &\equiv (\exists m \leq min(i,j), \\ &\quad (x_1, \ldots, x_{m-1}) = (y_1, \ldots, y_{m-1})) \\ &\quad \wedge (x_m < y_m) \\ ID_u \prec ID_v &\equiv (ID_u \prec_w ID_v) \vee (ID_u \prec_s ID_v) \end{aligned}$$

ID の拡張（ID の末尾に値を追加）後も関係 $\prec_s$ は成り立つが，ID の拡張後も関係 $\prec_w$ が成り立つとは限らないことに注意する．

定義 5.3 優先度の順位関係

$$\begin{aligned} &(ID_u, Distance_u) \gg (ID_v, Distance_v) \\ &\quad \equiv (ID_u \succ ID_v) \vee \\ &\qquad ((ID_u = ID_v) \wedge (Distance_u < Distance_v)) \end{aligned}$$

次にアルゴリズムについて述べる．

MAX-PRIORITY アルゴリズム（**図 5**）によって木の優先度の低い優先度をもつ木が高い優先度をもつ木と結合し，その優先度を更新する（[B]）．隣接するノードにより高い優先度をもつものが存在しない場合は自分がルートとなる（[C]）．これが局所的な初期化に値する．[A] は効率の面からつけられたものである．

MAXIMIZE – PRIORITY$_u$

Let l be one of $\{x|((ID_l, Distance_l) = max_{v \in Nbrs_u}(ID_v, Distance_v)\}$

if $(Parent_u = \text{nil})$ **and** $(ID_u \prec_w ID_l)$
then [A]
  while $ID_u \prec_w ID_l$
    Append – Entry$_u$()

if $(ID_l, Distance_l) \gg (ID_u, Distance_u)$
then [B]
  $ID_u \leftarrow ID_l$
  $Root_u \leftarrow Root_l$
  $Distance_u \leftarrow Distance_l + 1$
  $Parent_u \leftarrow l$
  $OtherTrees_u \leftarrow$ false
else [C]
  $Root_u \leftarrow UID_u$
  $Distance_u \leftarrow 0$
  $Parent_u \leftarrow$ nil
  $OtherTrees_u \leftarrow$ false

図5 $MAXIMIZE - PRIORITY_u$ アルゴリズム
Fig. 5 Action: $MAXIMIZE - PRIORITY_u$

このアルゴリズムを用いると，最終的には優先度の変化が止まり，全てのノードが，ルートか他のノードの子になる．これを極大木が存在した状態とみなしたいが，実際には各々のノードが優先度をそれ以上変化できない状態（[D]）であったとしても，同じ ID をもつルートが複数存在する可能性があり，必ずしも一つの木になっているとは限らない．

そこで，DETECT-TREES アルゴリズム（**図 6**）によって，ID の値は等しいがルートの異なる木をを検出し，ID の拡張を行い，優先度の変化が起こるようにする．ID の値が等しくルートの異なる木が存在する場合，そのことを変数 $OtherTrees$ を介して，各々の木のルートに伝える（[E]）．そして ID の等しい他の木が隣接するとき，ルートは ID の拡張を行う（[F]）．UID はルートごとに異なる値であるので．各々の木では優先度が異なるようになり，MAX-PRIORITY アルゴリズムによって木を結合することが可能になる．

よって，この 2 つのアルゴリズムを用いることによって，任意の状態から最終的に一つの極大木が生成された状態へと達することが可能となる．

定理 5.4 任意の状態から O(グラフGの直径) で適切状態に到達し，極大木が求まる．

5.2.3 コネクションを生成するホストの選択

求まった $Parent_u$ の値をもとに，ホスト間のコネクションを生成する．もし $(u, Parent_u) \in E$ であれば u が $Parent_u$ へのコネクションを生成し，$(u, Parent_u) \notin E$ であれば $Parent_u$ が u へのコネクションを生成する．

DETECT – TREES$_u$

if $(\forall v \in Nbrs_u, (ID_v = ID_u)$
  and $(|Distance_v - Distance_u| \leq 1))$
then [D]
  if $(\exists v \in$ **Children$_u$**$, OtherTrees_v = \text{true})$
    or $(\exists v \in Nbrs_u, Root_v \neq Root_u)$
  then [E]
    $OtherTrees_u \leftarrow$ tree

if $(Parent_u = \text{nil})$ **and** $(OtherTrees_u = \text{true})$
then [F]
  Append – Entry$_u$()

図6 $DETECT - TREES_u$ アルゴリズム
Fig. 6 Action: $DETECT - TREES_u$

Append – Entry$_u$()
  $ID_u \leftarrow ID_u : UID$

Children$_u$: $\{v|Parent_v = u\}$

図7 $Append - Entry_u(), Children_u$
Fig. 7 $Append - Entry_u(), Children_u$

DETECT – HOME$_u$

if (u is HomeNode)
then
  $Home_u \leftarrow u$
else if $(\exists v \in$ **Children$_u$** $\cup \{Parent_u\},$
    $(Home_v \neq \text{nil})$ **and** $(Home_v \neq u))$
then
  $Home_u \leftarrow v$
else
  $Home_u \leftarrow$ nil

図8 $DETECT - HOME_u$ アルゴリズム
Fig. 8 Action: $DETECT - HOME_u$

5.3 情報取得，ルーティングのアルゴリズム

生成された極大木を用いて通信を行い，ネットワークの位相をホームノードが把握する．各々のノードがホームノードへの経路を検出し，変化が生じる度に自分のホスト情報を送信する必要がある．

DETECT-HOME アルゴリズム（**図 8**）は極大木上でホームノードへの経路を検出する．

新たに $Home$ という変数を追加する．この変数の値は，ホームノードにおいては自己をさし，ホームノード以外では，子か親を表す．ノード $Home$ を中継して，ホームノードへ情報の送信を行う．

極大木自体にはサイクルが存在しないので，局所的にサイクルを防ぐことによって，$Home$ の値がサイクルを構成するのを防ぐことになる．これによって正しく経路の検出がなされることが保証される．

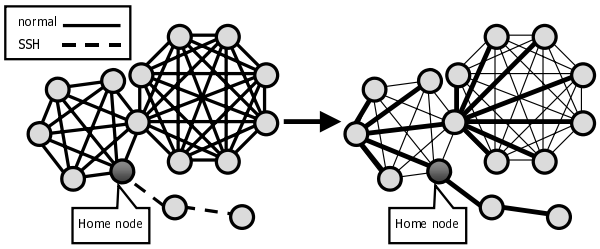

図 9 実験で使用したネットワークと構成された極大木
Fig. 9 experience: network and spanning tree

表 1 ジョブ投入によるオーバヘッド（単位: 秒）
Table 1 overhead of remote job submission (second)

	通常	1 台中継	2 台中継
globus-job-run	5.80	—	—
SSH	0.89	1.17	1.44
rsh	0.12	0.22	0.30
VPG	0.22	0.20	0.22

こうして，確保された経路を使って，自分のホストの情報を，更新される度にホームノードへと伝達する．ホームノードは求まった情報を結合することによってネットワークの位相を把握し，hop 数をもとに最短路を計算し，目的ホストへの経路を検出する．

6. 実　　験

6.1 実験内容

実験に使用した計算機は 15 台で，OS は Solaris, Linux，プロセッサは sparc，intel となっている．ネットワークは**図 9** のように 2 つのサブネットに分かれており，そのサブネット内でもアクセスの制限が存在する．また，シェルのプロトタイプを実装し，VPG と他の遠隔ジョブ投入ツールの比較を行った．rsh，SSH，globus-job-run☆を取り上げ，実行時間の短いジョブの投入時間を計測するを行うことによって，投入自体にかかるオーバヘッドを計測，比較した．VPG，rsh，SSH においては多段にホストを中継してのジョブ投入についても計測した．

6.2 実験結果

実行後，直径が 5 の極大木が選択された（図 9）．実際にどのような極大木が構成されるかはホストのニックネーム（UID），追加削除の操作によるホストの増減が影響する．極大木の構成にかかる時間は，隣接するホストからの情報取得間隔等に依存する．
またジョブ投入時間の計測結果は**表 1** のようになった．VPG は globus-job-run や SSH よりオーバヘッドが小さく，多段にホストを中継することによる影響も SSH と比較して小さい．
まとめると，VPG は必要最低限のコネクションを生成し，サブネットをまたいだ効率的なジョブ投入を実現することが示せた．

7. 結　　論

複数管理ドメインにまたがる多数の計算機を安全かつ簡便に利用可能にするための Grid 計算用のシェルを設計・実装した．そして実際に 15 台の計算機上で実験を行った．
今後の課題としては，

- 遠隔環境での実験
- 既存のシェル上での実装（シェルスクリプトによるバッチジョブ投入など）
- 利用者が前もって与えなければ行けない情報の削減（今現在ではそのホストが接続可能なホストのリストを与えなければいけない）．
- ユーザによる明示的なホストの指定なしの並列資源選択

などが挙げられる．

参 考 文 献

1) Foster, I. and Kesselman, C.: *The GRID: Blueprint for a New Computing Infrastructure*, Morgan kaufmann Publishers (1998).
2) Foster, I. and Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit, *Proc. the Workshop on Environments and Tools for Parallel Scientific Computing* (1996).
3) Tanaka, Y., Sato, M., Hirano, M., Nakada, H. and Sekiguchi, S.: Resource Manager for Globus-based Wide-area Cluster Computing, *1st IEEE International Workshop on Cluster Computing (IWCC'99)*, pp. 237–244 (1999).
4) http://www.ssh.com/
5) Scott, C., Wolfe, P. and Erwin, M.: (須田 隆久 訳, 歌代 和正 監訳) VPN 第 2 版, O'REILLY オライリージャパン (2000).
6) Afek, Y., Kutten, S. and Yung, M.: Memory-Efficient Self Stabilizing Protocols for General Networks, *4th Workshop on Distributed Algorithms* (1990).
7) Aggarwal, S. and Kutten, S.: Time optimal self-stabilizing spanning tree algorithms, *13th Conference on Foundations of Software Technology and Theoretical Computer Science* (1993).

☆ Globus の提供する遠隔ジョブ投入ツール