PluS 予約機構の Condor への適用

中田秀基[†]　竹房あつ子[†]　大久保克彦[†,††]
工藤知宏[†]　田中良夫[†]　関口智嗣[†]

複数資源の同時確保には，各資源が事前予約機構を備えていなければならない．我々は既存のローカルスケジューラに対して事前予約機構を付加する PluS 事前予約機構を開発している．PluS 事前予約機構は，2 つの動作モードを持つが，そのうちの 1 つであるキュー制御動作モードは，ローカルスケジューラが特定の機能を持つことを前提に，モジュール構成を改変すること無く 予約機構を付加することができる．このキュー制御動作モードは，特定のローカルスケジューラに依存しないよう 注意深く 設計されているが，これまでの実際の適用例は，Sun Grid Engine のみでキュー制御動作モードの汎用性は立証されていなかった．本稿では，キュー制御動作モードの汎用性を立証するべく，Sun Grid Engine とは全く 異なる基本設計に基づくローカルスケジューラ Condor に対して PluS の適用を行った．その結果，わずかなコード量で Condor への適応が可能なことを確認した．

Application of the PluS reservation Manager to Condor

Hidemoto Nakada ,[†] Atsuko Takefusa ,[†]
Katsuhiko Ookubo,[†,††] Tomohiro Kudoh ,[†] Yoshio Tanaka [†]
and Satoshi Sekiguchi [†]

For resource co-allocation it is essential to have advance reservation capabilities on each resoucces, such as computers and networks. We developped an advance reservation manager **PluS** for computational resources that works as a plug-in for existing local batch queuing schedulers. PluS in the queue-controling mode, which is one of the two operational modes supported, works from completely outside of the target batch queueing schedulers. Although the mode was implemented for Sun Grid Engine, we carefully designed the core login of the mode independent from any specific batch queueing schedulers, so that we can reuse the logic to other queueing schedulers. The goal of this paper is to confirm the portability of the mode. As a target queueing system, we employed **Condor**, which has totally different configuration from other queueing systems, including Sun Grid Engine. As the result, we confirmed that it is possible to work with Condor for PluS, and the required code size is relatively small.

1. はじめに

グリッドの目的の一つとして，ネットワークや計算機，その他の機器などの複数の資源を同時に確保することによる大規模な計算がある．複数資源を同時に確保する方法はいくつか考えられるが，もっとも簡単な方法の一つが，各資源に事前予約機構を与え，予約によって特定の時間帯における資源の同時確保を行う 方法である．我々は計算機資源の事前予約による確保のために，既存のローカルスケジューラに対して事前予約機構を付加する PluS 事前予約機構 [1] を開発している．PluS 事前予約機構は，スケジューリングモジュール置換動作モードと，キュー制御動作モードの 2 つの動作モードを持つ．後者は，対象となるローカルスケジューラが特定の機能を持つことを前提に，対象のモジュール構成を改変すること 無く 予約機構を付加することができる．このキュー制御動作モードは，特定のローカルスケジューラに依存しないよう 注意深く 設計されているが，これまでの実際の適用例は，Sun Grid Engine[2] のみでキュー制御動作モードの汎用性は立証されていなかった．

本稿では，キュー制御動作モードの汎用性を立証するべく，Sun Grid Engine とは全く 異なる基本設計に基づくローカルスケジューラ Condor[3),4)] に対して PluS の適用を行った．Condor にはジョブキューという 概念がなく，キューを制御する方法はそのまま適用できない．Condor のマッチメイキング機構を利用し，動的に実行計算機の設定を制御することで同等の機能を実現し，わずかなコード量で Condor への適応が可能なことを確認した．さらに，Condor に対する 予約

† 産業技術総合研究所　National Institute of Advanced Industrial Science and Technology (AIST)
†† 数理技研　SURIGIKEN Co., Ltd.

操作は実用的な時間で終了することを確認した．一方，PluS の提供するサービスプロバイダインターフェイスは，2 つの動作モードに対応するため煩雑な側面があり，改善の余地があることもわかった．

本稿の構成を以下に示す．2 節でベースとなる PluS 予約機構の概要を，3 節で Condor の概要を示す．4 節で PluS の Condor への対応の設計と実装について述べる．5 節で評価を行う．6 節はまとめである．

2. PluS の概要

PluS は，既存キューイングシステムである，TORQUE[5] および Grid Engine[2] に対してプラグインとして動作する事前予約機構である．キューイングシステムに対してプラグインする方法として，キューイングシステムの既存スケジューリングモジュールを完全に置換する方法 (スケジューラ置換動作モード) と，ジョブキューを外部から制御することで予約を実現する方法 (キュー制御動作モード) の 2 つをサポートしている [6]．予約操作に関して，2 相コミットを実現している点も特徴の一つである．また，予約受け入れポリシの記述を Condor の ClassAd を用いてサイト管理者に開放することで，各サイトのポリシに応じた運用を可能にしている [7]．

2.1 キュー制御動作モードの動作

キュー制御動作モードで運用する場合の PluS とキューイングシステムの関係を図 1 に示す．一般にキューイングシステムは，ヘッドノード及び実行計算機の 2 種類のノードによって構成される．実行計算機上ではジョブの実行を管理するデーモンが動作する．ヘッドノード上では，マスターデーモンとスケジューリングデーモンが稼働する．マスターデーモンは実行デーモンを管理すると同時に，ジョブキューを管理する．ユーザはマスターデーモンに対してジョブを投入する．マスターデーモンは，実行デーモンの情報とジョブキューの情報をスケジューリングモジュールに渡し，ジョブの配置決定を依頼する．

PluS 予約モジュールはマスターデーモンに対して制御コマンドを発行することによって事前予約を実現する．具体的には，各予約に対して，対応する専用のジョブキューを作成し，そのジョブキューおよびデフォルトで存在するジョブキューの活性を時系列にそって制御することによって，特定時間帯にそのジョブキューが排他的に実行計算機を占有することを実現する．

PluS は，ユーザに対して一連の予約関連コマンド群を提供する．予約依頼コマンドは特定の予約 ID をユーザに返却する．ユーザはその予約 ID を指定してジョブをキューイングシステムに対してサブミットする．

2.2 キュー制御動作モードの可搬性

キュー制御動作モードは Sun Grid Engine との連携を念頭において設計されたが，PluS が想定する機能を満たしているキューイングシステムに対して可搬となるように配慮がなされている．PluS は Java で記述されており，SPI(サービス・プロバイダ・インターフェイス) としていくつかの Java インターフェイスや抽象クラスを定義している．これらに対して具体的な実装を与えるだけで PluS のキュー制御による事前予約機能を実現することができる．PluS の SPI で用意されているインターフェイスと抽象クラスとその意味を表 1 に示す．

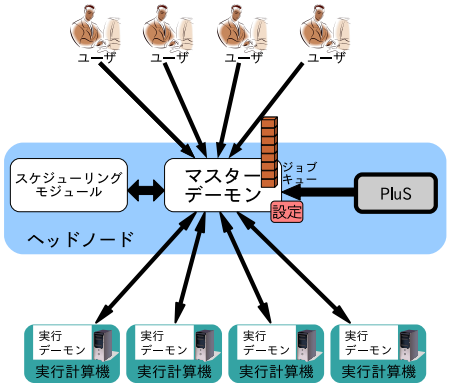

図 1 PluS と通常のキューイングシステム

表 1 PluS のサービスプロバイダインターフェイス

名前	種別	意味
JobID	I/F	ジョブ ID
JobStatus	I/F	各ジョブの状態
NodeStatus	I/F	各実行計算機の状態
QueueStatus	I/F	予約を示すジョブキューの状態
QInstanceStatus	I/F	予約を示すジョブキューの各実行計算機上の状態
MainServer	I/F	プログラムのエントリポイント
ReserveInfo	A/C	各予約を表す
ReserveManager	A/C	予約時間開始時，予約時間終了時の動作を記述する
ServerStatus	I/F	PluS モジュールそのものの状態を表す

* I/F - インターフェイス，A/C - 抽象クラス

PluS が事前予約機能を提供するために想定するキューイングシステムの機能は以下のとおりである．

- 特定のノード集合に対して，特定のユーザ集合がサブミットした特定のジョブ集合のみを実行するように，動的に設定可能であること．

キューイングシステムの多くでは，この機能は「ジョブキュー」として実現されている．ジョブキューは，特定のノード集合とユーザ集合に割り当てられる．そのジョブキューを指定して投入されたジョブは，特定のノード集合上で実行される．すべてのキューイングシステムのジョブキューがこの条件を満たすわけではないことに注意が必要である．例えば TORQUE はジョブキュー機能をサポートしているが，ユーザ集合に対して束縛することができないため，PluS のキュー

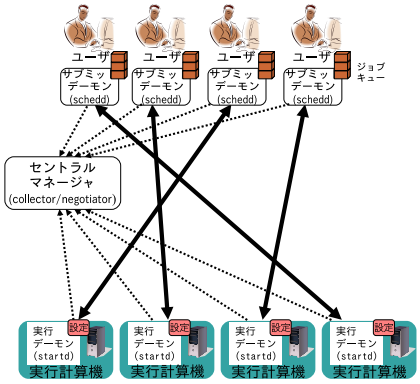

図 2 Condor の概要

制御動作モードを利用することはできない．

3. Condor の概要

Condor は、米国ウィスコンシン大学のグループが開発したハイスループットコンピューティングを指向したキューイングシステムである．当初はキャンパス内の遊休計算機を有効利用することを目的としていたが，現在ではグリッド上のメタスケジューラとしても広く使用されている．中心となる制御機構と設定点を置かず，各実行計算機が個別の権限で個別の設定を行う点が特徴である．これは Condor が各ユーザが持ち寄った資源を利用する有休計算機利用を指向しているためである．

3.1 Condor の構造

Condor は，サブミット計算機，セントラルマネージャ，実行計算機の，3 種類のノードから構成される (図 2)．サブミット計算機は典型的には，各ユーザの計算機であり，ユーザのジョブキューを管理する．セントラルマネージャ上ではネゴシエーションのためのモジュールが稼働し，後述するマッチメイキングにより実行すべきジョブと計算機の対応を決定する．実行計算機は，サブミット計算機からジョブを受け取り，実行する．このときジョブの実行は，セントラルマネージャを介さず，サブミット計算機と実行計算機の両者が直接通信して行うことに注意が必要である．

3.2 ClassAd とマッチメイキング

マッチメイキングは，ジョブを実行する計算機を決定する機構である．マッチメイキングは，ClassAd[8] と呼ばれる一種のポリシ言語に基づいて行われる．

Condor においては各ジョブと実行計算機がそれぞれ ClassAd を持つ．ジョブの ClassAd には，ジョブの状態や転送すべきファイルなどのジョブの属性の他に，そのジョブを実行するためのリソースへの要求が記述される．要求としては，実行計算機のアーキテクチャや OS，メモリ量などを記述することができる．この要求は`Requirements`と呼ばれる属性で記述される．例として，アーキテクチャとして Intel x86 を，OS として Linux を要求するジョブの`Requirements`を示す．

```
Requirements = (Arch == "INTEL") &&
               (OpSys == "LINUX")
```

一方，実行計算機の ClassAd には，実行計算機のアーキテクチャ，OS，メモリ量などの属性のほか，どのようなジョブの実行を許すかを記述することができる．例えば，特定のユーザのジョブや特定の属性を持つジョブのみの実行を許すように実行計算機を設定することが可能である．このジョブの開始条件は`Start`で記述する．たとえばユーザ`foo`に対してだけ実行を許す場合，`Start`式を以下のようにすればよい．

```
Start = (Owner == "foo")
```

マッチメイキングは，ジョブの ClassAd と実行計算機の ClassAd を付き合わせて，実行可能なペアを選択する作業である．各サブミット計算機上のサブミットデーモン (schedd) は，定期的にジョブキュー内のジョブの ClassAd をセントラルマネージャ上のデータ保持デーモン (collector) に送信する．同様に，各実行計算機上の実行デーモン (startd) は定期的に実行計算機の ClassAd を実行デーモンに転送する．セントラルマネージャ上のネゴシエーションデーモン (negotiator) は，定期的に起動して collector からジョブと実行計算機の ClassAd を受け取り，両者を付き合わせて，双方の条件が互いにマッチするジョブと計算機の組み合わせを構成し，該当する実行デーモン，およびサブミットデーモンに知らせる．

特徴的なのは，これらの条件に記述する項目を自由に拡張することが出来る点である．例えば，Condor 自身はユーザグループという概念を持たないが，設定ファイルでユーザグループを定義し，そのユーザグループに対して特定の計算資源を割り当てる，といった制御が容易に可能である．

4. システムの設計と実装

4.1 設　　計

前節で述べたとおり，Condor は通常のバッチキューイングシステムとまったく異なる構造を持つ．まず，いわゆるジョブキューという概念が存在しない．基本的にすべてのジョブは同一空間にプールされる．また，設定を集中して制御するマスターデーモンも存在しない．設定は，各実行マシン上の実行デーモンに分散して存在しているため，個別に制御しなければならない．

我々は，ジョブキューの代替として，マッチメイキング機構による実行ジョブの選別機構を用いた．機構への要請は，下記の 2 点である．

(1) 特定のユーザ群の特定のジョブ群が，特定のノー

ド群で動き，それ以外のノード群では動かないこと

(2) 特定のジョブ群以外のジョブが，特定のノード群で動かないこと

各実行ノードの ClassAd とジョブの ClassAd を改変することによって，これらを実現した．

まず，上記の 1 を実現するためには，実行計算機が特殊な属性を定義し，ジョブがその属性を持つ計算機を要求するようにすればよい．属性としては，PlusResourceFor を予約 ID に束縛することとした，実行計算機側では，下記の属性を定義する．

```
PlusResourceFor="RID"
```

これに対して，ジョブには下記の Requirements を定義する．

```
Requirements = (PlusResourceFor == "RID")
```

ここで RID は個々の予約を表すユニークな識別子である．ジョブが Requirements として，PluSResourceFor 属性を持ち，それが予約 ID と一致している実行計算機を指定しているため，該当する計算機以外でジョブが動くことはない．

上述の 2 を実現するためにはこれだけでは十分ではない．実行計算機の側では実行対象のジョブを制約していないため，予約ジョブ以外のジョブが動いてしまう可能性がある．これを回避するためには，逆にジョブに特定の属性を持たせ，実行計算機の側でその属性を要求するようにすればよい．このための属性として ResvID を用いた．ジョブには下記を指定する．

```
ResvID = "RID"
```

これに対して，実行計算機では下記のように Start 式を定義する．

```
Start=(ResvID=="RID")
```

これによって特定の属性値をもつジョブ意外は実行されなくなる．しかしこれでは十分ではない．ジョブの属性値はユーザが自由に設定することができるため，本来その予約スロットを利用する権限の無いユーザが，この属性値を詐称することで予約スロットを利用できてしまう可能性がある．これを防ぐため，予約スロットを利用する権限をもつユーザを Start 式内で指定する．

```
Start = (ResvID=="RID") &&
        (Owner=="USR1" || Owner="USR2")
```

予約ジョブと割り当てられた資源の ClassAd の概要を図 3 に示す．対応ができていることが分かる．

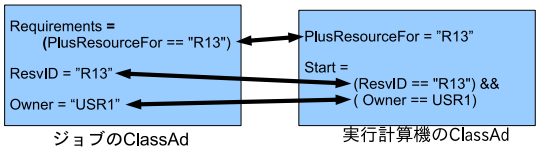

図 3 予約ジョブと割り当てられた資源の ClassAd

Condor に PluS を適用したシステムの動作は以下の通りとなる．

(1) ユーザが予約を依頼する．PluS はデータベース内の予約テーブルを参照し，予約が受け入れ可能かどうかを判断する．可能な場合には，ノードと予約識別子を割り当て，予約テーブルに記録し，予約識別子をユーザに対して返却する．この際，Condor に対してはなにも動作を行わない．

(2) ユーザは Condor に対してジョブをサブミットする．この際，上述のように Requirements 属性と ResvID 属性を付加する．Requirements 属性を満たす実行計算機がないため，この時点では実行されない．

(3) 予約開始時間になると，PluS は実行計算機の ClassAd の Start 属性と PlusResourceFor 属性を変更する．これによって，ユーザが予約識別子を指定して投入したジョブの実行が可能になり，ジョブの実行が始まる．

(4) 予約終了時間になると，PluS は実行計算機の ClassAd を変更して復旧する．

4.2 実　　装

2.2 で示した SPI のインターフェイス，抽象クラス群に実装を与えることで，Condor 向け PluS の実装を行った．SPI の定義するメソッドは，Condor の状態を取得するものと，Condor に対して操作を行うものに大別することができる．

4.2.1 Condor の状態の取得

Condor の状態の取得は，Condor の提供する condor_status および condor_q で行った．これらのコマンドには，状態を ClassAd の XML 形式で出力するオプションが用意されている．さらに，ClassAd には操作するための Java ライブラリが用意されている．これらを用いることで非常に容易に，Condor の状態を取得することができる．

一点問題になるのはジョブキューの状態の取得である．PluS はジョブキューの存在を想定しており，ジョブキューの状態を取得して操作を行おうとする．しかし Condor にはジョブキューがないため，何らかの方法でジョブキューに相当する状態を作り上げなければならない．われわれは，データベース内の予約テーブルから取り出した情報と，実行計算機に設定された属性から，PluS が期待するジョブキューの状態に相当する情報を生成した．

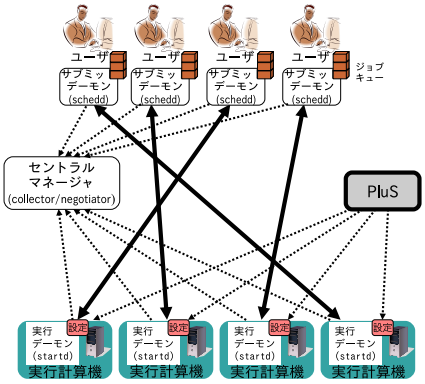

図 4　PluS の Condor への適用

4.2.2　Condor の制御

制御の実際の動作は Python によるスクリプトで実現した．これは主にデバッグの容易さを重視しての選択であり，将来的にこの部分も Java で記述しなおすことも検討している．

Condor には Sun Grid Engine におけるマスターデーモンのような単一の設定変更点がなく，各実行計算機上の実行デーモン startd に対して個別に設定変更を行わなければならない (図 4)．Python スクリプトは Condor の提供する設定情報更新コマンド `condor_config_val` および`condor_reconfig` を利用して，startd の ClassAd を更新する．さらに，非予約ジョブが実行中であった場合には，`condor_vacate` コマンドを用いて，そのジョブの実行を停止し，予約ジョブが実行可能な状態にする．この際，ノード数の増大に伴って設定更新時間が増大することを避けるために各実行計算機に対して個別のスレッドを用いて，設定更新を行う．

4.2.3　ジョブ投入ラッパ

4.1 で述べたように，ジョブ投入時には，ジョブの属性値として`Requirements` に`PluSResourceFor` を追加し，さらに，`ResvID` を定義する必要がある．これらはサブミットファイルを編集することでも実現できるが，これをユーザが毎回行うことは煩雑である．このため，自動的にこれらの属性の設定を行うジョブ投入補助シェルスクリプト `plus_condor_submit` を用意した．

このスクリプトは，`condor_submit` のラッパとして機能し，引数として予約識別子をとる．まず，指定されたサブミットファイルを利用してジョブを，実行されることのないホールド状態で投入する．つぎに，`condor_submit` の出力を解析してジョブ ID を取得し，このジョブ ID を用いてこのジョブの ClassAd を `condor_qedit` コマンドで改変する．最後に，このジョブをリリースして実行可能にする．

表 2　予約時間開始時の設定変更コスト (s)

ノード数	平均	最大	最小	偏差
1	7.08	7.58	6.77	0.29
2	7.80	8.35	7.26	0.34
3	8.00	8.81	7.38	0.43
4	8.08	8.82	7.63	0.38
5	8.40	9.02	7.80	0.44
6	8.58	9.22	7.89	0.45
7	8.73	9.26	8.32	0.33

表 3　予約時間終了時の設定変更コスト (s)

ノード数	平均	最大	最小	偏差
1	7.24	7.96	6.82	0.32
2	7.60	8.12	6.96	0.42
3	7.50	8.28	6.94	0.39
4	7.65	8.16	7.17	0.37
5	7.75	8.28	7.13	0.47
6	7.87	8.43	7.15	0.48
7	7.76	8.47	7.06	0.63

5.　評　　価

5.1　実行時間とノード数に対するスケーラビリティ

前節で述べたとおり，Condor では各実行デーモンに対して設定を行わなければならない．この際，設定ファイルを再読み込みさせる必要があり，これが設定変更の実行時間に大きな影響を与える可能性がある．また，各実行計算機に対して個別に設定を行わなければならないため，実行計算機の数が増加するにつれて設定変更のコストが増大する可能性がある．

これらの問題について確認するために簡単な実験を行った．実験環境は Giga bit Ether で接続されたクラスタ計算機で，各ノードは 2.8GHz の Intel Pentium4 Xeon プロセッサで，1Gbyte のメモリを搭載している．OS は CentOS 4 である．Condor はバージョン 6.9.3 を用いた．予約対象となる実行計算機の台数を変化させ，予約時間開始時と終了時の設定変更に要した時間を計測した．計測は，予約管理システムを模した Java プログラムから，制御用の Python プログラムを起動して行った．計測はそれぞれ 10 回行っている．

表 2,3 に，予約時間開始時，終了時の設定変更に要した時間を，平均値のみをプロットしたグラフを図 5 に示す．実行計算機数によらず，7 秒から 9 秒程度で設定変更が完了していることが分かる．コストがノード数に比例していないのは，python による制御プログラムが各実行計算機に対する制御をスレッドで並列に行っているためである．

しかし，予約実行開始時の設定変更コストは実行計算機数の増加に応じて若干増大している．数十台程度であればそれほど問題にならないと考えられるが，さらに実行計算機を増やす場合には，PluS 側から複数の python プログラムを起動するなどの方法でコスト

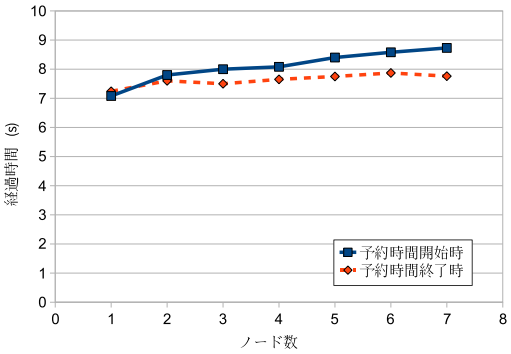

図 5 予約時間開始時，終了時のコスト

表 4 コード行数

種別	ファイル数	行数計
Java	10	1068
python	2	324
sh	1	75
計	13	1467
SGE 版行数総計	-	約 9800

の増大を避ける必要がある．

PluS による事前予約を Sun Grid Engine に対して行う場合，ジョブキュー設定の変更は，同様の環境で 1 秒以内で完了する．この差は大きいが，設定変更はユーザコマンドの投入時ではなく，予約時間開始終了時に行われるためユーザコマンドのレスポンスタイムには影響を与えない．したがって，システムの挙動としては問題ないと考える．

5.2 コード行数の評価

今回の実装に必要となったコード行数を表 4 に示す．Java によるソースコード量は 1000 行程度，python による補助コマンドが 300 行程度となっている．sh となっているのは，4.2.3 で述べたサブミットコマンドのラッパである．

最下段に示した値は，Sun Grid Engine 版の行数の総計である．この行数には，SPI を実装したコードだけでなく，予約を制御するためのすべてのコードが含まれているため直接比較することはできないが，Condor 対応版の実相が比較的低いコストで実装できていることが分かる．

6. おわりに

本稿では，PluS 予約管理システムの，キュー制御動作モードの汎用性を立証するべく，Sun Grid Engine とは全く異なる基本設計に基づくローカルスケジューラ Condor に対して PluS の適用を行った．その結果，わずかなコード量で Condor への適応が可能なことを確認した．

今後の課題は以下の通りである．

- 2.2 で述べた通り，PluS は各キューイングシステムに対応するための SPI を提供している．しかし，この SPI はキュー制御動作モードとスケジューラ置換動作モードで共有されているため，どちらか一方を実装する場合，他方のために提供されているメソッドは不要であるにもかかわらず，ダミーを記述しなければならない．これはプログラマにとって大きな負担となる．キュー制御動作モードとスケジューラ置換動作モードの SPI を明確に分離することで，より実装を容易にする必要がある．
- 今回の実装では，Condor の設定の制御に関するセキュリティ問題に関しては考慮していない．このため，PluS が稼働するノードからはだれでも Condor の設定の制御が出来てしまう．Condor は本来強固なセキュリティレイヤを持っており，この問題は純粋に Condor の設定の問題である．今後，セキュリティの設定を検討する．

謝　辞

Condor の詳細に関してご教示いただいたウィスコンシン大学 Condor チーム Jaime Fry 氏に感謝する．

本研究の一部は，文部科学省科学技術振興調整費「グリッド技術による光パス網提供方式の開発」による．

参　考　文　献

1) Nakada, H., Takefusa, A., Ookubo, K., Kishimoto, M., Kudoh, T., Tanaka, Y. and Sekiguchi, S.: Design and Implementation of a Local Scheduling System with Advance Reservation for Co-allocation on the Grid, *Proceedings of CIT2006* (2006).
2) Grid Engine. http://gridengine.sunsource.net.
3) Condor. http://www.cs.wisc.edu/condor/.
4) Raman, R., Livny, M. and Solomon, M.: Matchmaking: Distributed Resource Management for High Throughput Computing, *Proc. of HPDC-7* (1998).
5) TORQUE Resource Manager. http://www.clusterresources.com/pages/products/torque-resource-manager.php.
6) Nakada, H., Takefusa, A., Ookubo, K., Kudoh, T., Tanaka, Y. and Sekiguchi, S.: An Advance Reservation-Based Computation Resource Manager for Global Scheduling, *Third Workshop on Grid Computing and Applications*, pp. 3–14 (2007).
7) 中田秀基, 竹房あつ子, 大久保克彦, 工藤知宏, 田中良夫, 関口智嗣: 事前予約機構のポリシ記述による制御, 情報処理学会研究報告 2007-HPC-110, pp. 19–24 (2007).
8) ClassAd. http://www.cs.wisc.edu/condor/classad/.