

事前予約機構のポリシー記述による制御

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[†]

グリッド上の複数サイトにまたがるジョブの実行を実現する方法として、事前予約による同時確保がある。事前予約を実現するローカルスケジューラは、いくつか提案されているものの、事前予約ジョブと通常のジョブを共存させるための適切なポリシーは明らかになっていない。われわれは、事前予約ジョブと通常ジョブ間のポリシーは、各サイトのローカルスケジューラに内蔵されるべきではなく、管理者が設定すべき事項であると考え、実装中の PluS 事前予約機構に、管理者によるポリシー記述機能を組み込んだ。ポリシー記述には、Condor プロジェクトで用いられている ClassAd を用いた。我々は、1) ClassAd による記述力は管理ポリシーを記述するのに十分であること、2) PluS の提供する情報により有効な管理ポリシーが記述できること、3) ClassAd によるポリシー解釈のオーバーヘッドは十分小さいことを確認した。

Policy-based Control of Advance Reservation

HIDEMOTO NAKADA,[†] ATSUKO TAKEFUSA,[†]
KATSUHIKO OOKUBO,^{†,††} TOMOHIRO KUDOH,[†] YOSHIO TANAKA[†]
and SATOSHI SEKIGUCHI[†]

While advance reservation is the most promising way to enable co-allocation of distributed resources over the Grid, and there are several local schedulers that provide such a capability, it is still an open problem that how to setup 'policies' for reservation, such as, 'should I accept this reservation request?'. We claim that such a policy should not be embedded in the local schedulers, but should be able to be set by each site administrator. We implemented a policy evaluation capability in our PluS reservation manager to allow administrators to setup their own policy. As the policy language, we employed ClassAd which is developed as a part of the Condor Project. We confirmed that, 1) ClassAd is powerful enough to describe policies 2) Information provided by the PluS reservation manager is enough to describe meaningful policies 3) policy evaluation overhead is acceptable.

1. はじめに

グリッドの目的のひとつである複数サイトにまたがるジョブの実行を実現する方法として、事前予約による複数サイト上資源の同時確保がある。このためには、事前予約を実現するローカルスケジューラが必要となり、われわれの PluS を含めていくつかの事前予約機構が提案されている^{1),2)}。ここで問題となるのは、ユーザ間のフェアシェアと予約ジョブと非予約ジョブの共存である。通常の非予約ジョブのみを対象とするローカルスケジューラでは、FCFS (First Comes First Served) ベースのスケジューリングに加え、なんらかのプライオリティとフェアシェアが実現されており、複数のユーザのジョブがバランスをもって実行

される枠組が整えられている。しかし、事前予約ジョブに関してはこの種の枠組があるとはいえ、特に非予約ジョブと混在した場合のポリシーについては明確になっていない。このため、現存する事前予約機構の多くは過度に保守的なポリシーをとっている。例えば、PBS Professional では、非予約ジョブがキューイングされていないノードに対してだけ予約ジョブを割り当てる。Catalina¹⁾ は、全てのジョブを予約ジョブ扱いとしてノードに事前配置した上で、空いたノードと時間帯があれば、予約ジョブを受け付ける。しかし一般に、実運用されているクラスタの多くは、稼働率 100% に近く、このようなポリシーでは事前予約を実際にいれて複数資源の同時確保を行うことはほとんど不可能である。これに対して、われわれが過去に提案した PluS 予約機構^{3),4)} では、事前予約ジョブを最優先したポリシーを採用した。このポリシーでは、予約は他の予約に干渉しない限り必ず成功し、予約開始時刻には実行中の非予約ジョブをプリエンプトして予約ジョブを実行する。

[†] 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.

このポリシーは同時確保を最優先として考える場合には有効であったが、通常のクラスタの運用ポリシーとしては問題がある。また、ユーザ間のフェアシェア問題に関してはなにも解決できていない。

われわれは、事前予約ジョブと通常ジョブ間のポリシーは、事前予約システムに固定的に内蔵されるべきではなく、管理者が設定すべき事項であると考え。これを実現するためには、なんらかのポリシー記述言語を予約システムに内蔵し、管理者にポリシーの記述を許す必要がある。

われわれは、実装中の PluS 事前予約機構に、ポリシー解釈機能を組み込んだ。ポリシー記述言語としては、Condor プロジェクト^{5),6)}で開発され、Condor 内部のみならずいくつかのプロジェクトで使用されている ClassAd^{7),8)}を用いた。これによって、サイト管理者は、あるジョブ予約リクエストに対して任意のポリシーを定義し、運用することができる。

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 ClassAd の概要を示す。4節で ClassAd の PluS への組み込みの詳細を述べる。5節で、ClassAd を用いたポリシー記述の例を示す。6節では、ClassAd を用いることによるオーバヘッドを評価する。7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、TORQUE⁹⁾ および Grid Engine¹⁰⁾ 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と、キューを外部から制御することで予約を実現する方法の2つをサポートしている。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である。

キュー操作で運用する場合の PluS の構造を図1に示す。灰色のモジュールは、キューイングシステムが提供するモジュールで、中央上部の「予約管理モジュール」が PluS の提供するモジュールである。

PluS は予約操作コマンドをユーザに提供する。キュー操作では、個々の予約は個別のキューとして実現される。ユーザは、コマンドを利用して予約 ID を取得し、予約 ID を付記してジョブ投入を行う。通常のジョブ操作コマンドがキューイングシステムのマスタモジュールで処理されるのに対して、予約操作コマンドは、PluS の本体であるモジュールによって処理される。

PluS は、予約受けつけ時および予約開始/終了時刻に、一連のキュー操作コマンドをマスタモジュールに対して発行し、ジョブキューの作成、活性化、非活性化、破棄などの操作を行う。

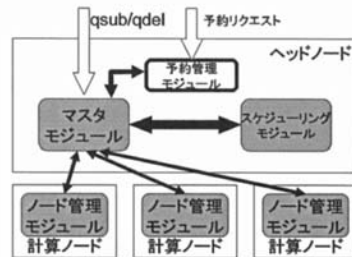


図1 PluS の構造

3. ClassAd の概要

3.1 ClassAd 言語

本節では、ポリシー記述に用いた ClassAd について述べる。ClassAd は、新聞などに求職、求人などに分類されて掲載される数行の広告を意味する Classified Advertisement の意である。Condor では、リソースやジョブがそれぞれの属性や要請を広告し、Negotiator と呼ばれるモジュールが、この情報を用いてリソースとジョブを引き合わせることによってジョブに対するリソースの割り当てを行っている。gLite^{11),12)} も同様に、ClassAd を用いたリソース割り当てを行う。

言語としての ClassAd は以下の特徴を持つ。

- 逐次的ではなく宣言的な記述
- 副作用がない
- 評価にかかる時間が、式の長さに対して線形

ClassAd は動的にストリクトな型付けを行う。基本型には、整数型、実数型、文字列、真偽値、時刻、相対時間が用意されており、この他に、エラーを表す *Error*、定義不能を示す *Undefined* がある。データ構造の型として、レコード型とリスト型がある。レコード型はキーワードと値のペアを格納する構造である。リスト型は任意の長さの値を列挙の格納する構造である。レコード型もリスト型も任意回数ネストすることができる。

図2に簡単な ClassAd の例を示す。全体を囲んだ大括弧 ('()') が、レコード型を示している。b は、ネストしたレコードの定義である。c の定義の右辺に表れている中括弧 ('[]') はリスト型を示している。d の右辺はリスト型の要素に対する参照である。e はセレクションと呼ばれる操作で、レコードの内部を参照している。f の右辺は 'isInteger' 関数を呼び出す関数呼出となっている。このデータ構造の抽象構文木を図3に示す。

ClassAd の評価は、各レコードをスコープとして変数の値を決めることで行う。スコープ (レコード) がネストしている場合は内側のスコープを優先して値を決定する。図2上段に示した ClassAd を評価した結果を図2下段に示す。b.q は a を参照しているが、こ

```

[
  a = 1;
  b = [a = 2; q = a;]
  c = {a, "xxx"}; // ClassAd 例
  d = c[0];
  e = b.q;
  f = isInteger(a);
]
-----
[
  a = 1;
  b = [a = 2; q = 2;]
  c = {1, "xxx"}; // 評価結果
  d = 1;
  e = 2;
  f = true;
]

```

図 2 ClassAd の例と評価結果

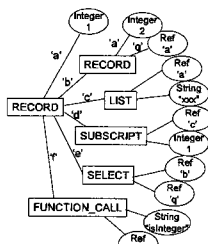


図 3 抽象木

のaは、外側のレコードではなく、内側のレコードを参照するため、1ではなく2と評価されている。

3.2 ClassAd 処理系

ClassAdの処理系は、C++によるものとJavaによるものが、Condor チームにより公開され、メンテナンスされている。これらの処理系には、独自記法およびXML記法のパーザ、アンパーザが含まれている。処理系はライブラリとして提供されているため、容易に他のシステムに組み込んで利用することができる。

ClassAd 処理系には、基本的な関数が組み込み関数として含まれている。さらにユーザが独自に関数を定義して組み込むことも可能である。この場合、関数は処理系が記述されている言語(C++もしくはJava)で記述する。

4. システムの設計と実装

4.1 要 請

以下に事前予約システムのポリシー記述に対する要請をまとめる。

- ユーザ間のフェアシェアを実現できる
- 事前予約ジョブと非予約ジョブの間のフェアシェアを実現できる
- 管理者が自由に設定できる
- ポリシ記述が直感的で再利用可能である
- ポリシの評価時間が、レスポンスタイムに大きな影響を与えない

4.2 設 計

上記の要請を満たすために我々は下記の設計方針を設定した。

- ポリシ判断に必要な様々な情報を可能な限り提供する
- 記述力が高い宣言的言語を用いる

前者は、フェアシェアを実現するためである。フェアシェアを実現するためには、現在のジョブの情報を可能な限りポリシー判定部に提供し、判断の材料とする必要がある。後者は逐次的な言語では、記述によっては、極端に評価時間がかかってしまう場合があり、評価時間の見積りができなくなってしまう可能性があるためである。

われわれはClassAdをポリシー言語に用いた。ClassAdは、評価時間が式の長さの線形時間に収まることが保証されている。またClassAdではレコードエントリの定義が、ユーザ定義関数的に機能するため、比較の見通しがよく再利用可能なポリシーの記述が可能である。

4.3 実 装

ClassAdによるポリシー記述を可能にするために、PluS事前予約機構を改良した。管理者は、事前にClassAdを利用して予約受け入れポリシーをファイルに記述しておく。ポリシーは、単一のノードに対するリクエストを受け入れるかどうかを真偽値で表現する式として記述する。

事前予約モジュールは、ユーザからの事前予約リクエストを受領すると、管理者の設定したポリシーと、現在のノードおよびジョブ状態とリクエストを勘案して、その予約リクエストを受け入れるかどうかを判断する。具体的には、管理者の記述したポリシーClassAdをファイルからロードし、これに、現在のノードおよびジョブの状態と、リクエストの情報をClassAdレコードで表現したものをマージして、レコード中のPLUS_RESERVABLE_NODEを評価する。リクエストおよび状態を表現するためにPluSが提供する変数を表1に示す。

ClassAdによる判定は、個々の候補ノードに対して行う。まず、その予約時間帯に使用可能なノードのリストを作成し、その中のノードを、1つつ可否の判定対象となるPLUS_CANDIDATE_NODEに設定し、ポリシーClassAdを繰り返し評価する。このループは、リクエストされたノード数が予約可能と判定された時点で終了する。

各ノード情報は、ClassAdのレコードとして表現される。各レコードのメンバを表2に示す。ノード情報には、実行中のジョブを示すレコードのリストも含まれる。このレコードリストのメンバを表3に示す。ジョブ情報には、ジョブのオーナーや開始時刻、walltimeなどの様々な情報が含まれている。

| | 名前 | 型 | 意味 |
|-----------|----------------------|-------------|--------------------|
| 予約リクエスト情報 | PLUS_RSV_OWNER | 文字列 | 予約をリクエストしたユーザ |
| | PLUS_RSV_START | 絶対時刻 | 予約開始時刻 |
| | PLUS_RSV_END | 絶対時刻 | 予約終了時刻 |
| 内部状態 | PLUS_ALL_NODES | ノードレコードのリスト | すべての実行ノードの情報 |
| 対象ノード | PLUS_CANDIDATE_NODE | ノードレコード | 予約可能かどうかを判定する対象ノード |
| | PLUS_ALLOCATED_NODES | ノードレコードのリスト | すでに予約対象となったノードのリスト |

| 名前 | 型 | 意味 |
|----------|-------------|--------------|
| name | 文字列 | ノードの名前 |
| isAlive | 真偽値 | 実行デーモン稼働状況 |
| loadavg | 実数 | ノードのロードアベレージ |
| nRunJobs | 整数 | 実行中のジョブの数 |
| jobs | ジョブレコードのリスト | 実行中のジョブ情報リスト |

| 名前 | 型 | 意味 |
|-----------|------|---------------------------------------------------|
| id | 文字列 | ジョブ ID |
| owner | 文字列 | ジョブオーナー名 |
| state | 文字列 | ジョブ状態 (Queued, Running, Exiting, Held, Suspended) |
| priority | 整数 | プライオリティ |
| startTime | 絶対時刻 | 開始時刻 |
| wallTime | 相対時間 | walltime |

```
UnusableNodes = {"unusableA", "unusableB"};

PLUS_NODE_RESERVABLE =
(PLUS_CANDIDATE_NODE.nRunJobs == 0 &&
 PLUS_CANDIDATE_NODE.isAlive &&
 PLUS_CANDIDATE_NODE.loadavg <= 0.3 &&
 !member(PLUS_CANDIDATE_NODE.name,
 UnusableNodes));
```

図 4 簡単なポリシーの例

5. ポリシ記述例

PluS のポリシー記述力を示すために、いくつかのポリシーのサンプルを示す。

5.1 簡単なポリシー

図 4 に、簡単なポリシーの例を示す。この例では、使用不能なノードが 2 つある場合に ("unusableA", "unusableB"), それ以外のノードから、稼働中で、実行中のジョブがなく、しかもロードアベレージが 0.3 以下のノードを探して利用するというポリシーを記述している。

5.2 比較的複雑なポリシーの例

図 5 に、ユーザのプライオリティと、予約開始時刻までの時間を考慮にいられた予約ポリシーの例を示す。このポリシーでは、VIP ユーザ、一般ユーザ、低優先度ユーザの 3 種類のユーザを仮定している。VIP ユーザ

```
MaxPeriod = relTime("00:30:00");
MinPeriod = relTime("02:00:00");
LimitPeriod = relTime("7d");
MaxReserveDuration = relTime("2d");
VIPs = {"userA", "userB", "userC"};
Users = {"userX"};
VIPRatio = 100.0;
UsersMaxRatio = 90.0;
UsersMinRatio = 50.0;
OthersMaxRatio = 50.0;
OthersMinRatio = 30.0;

MaxRatio = member(PLUS_RSV_OWNER, Users) ?
    UsersMaxRatio :
    OthersMaxRatio;
MinRatio = member(PLUS_RSV_OWNER, Users) ?
    UsersMinRatio :
    OthersMinRatio;

now = absTime(time());
prev = PLUS_RSV_START - now;
duration = PLUS_RSV_END - PLUS_RSV_START;
ratioFunc = linear(prev, MaxPeriod, MaxRatio,
    MinPeriod, MinRatio);

rsvRatio =
    (prev <= relTime("0") ||
     LimitPeriod <= prev ||
     duration >= MaxReserveDuration) ? 0 :
    member(PLUS_RSV_OWNER, VIPs) ? VIPRatio :
    (prev <= MaxPeriod) ? MaxRatio :
    (prev >= MinPeriod) ? MinRatio : ratioFunc;

nAllocate = size(PLUS_ALLOCATED_NODES) + 1;
nAllocatable =
    size(PLUS_ALL_NODES) * 0.01 * rsvRatio;
PLUS_NODE_RESERVABLE = (nAllocate <= nAllocatable);
```

図 5 予約時刻までの時間とユーザのプライオリティを考慮したポリシーの例

は空いた計算資源があれば常に予約を入れることができるが、一般ユーザは資源の 5 割まで、低優先度ユーザは資源の 3 割までしか利用できない。これは、後から VIP ユーザが予約を希望する場合に備えて、予約スロットを確保するためである。ただし、予約時刻が直近である場合には、VIP ユーザが予約を希望する可能性が低くなったと考え、一般ユーザ、低優先ユーザの予約可能割合を増やす。これは、ホテルなどが VIP 用に確保している部屋を当日のたびこみ客に開放するのと同じ考えかたである。この例では、予約時刻の 2 時間前から、30 分前にかけて線形に予約可能割合を増加させている。つまり、3 時間前に予約を試みて失敗しても、30 分前になれば予約が成功する可能性があるということである。図 6 に、各ユーザの予約可能割合の変化を示す。

5.3 ユーザの予約状況を参照するポリシー

図 7 に、ユーザの予約状況に応じて予約の可否を決定するポリシーを示す。この例では、現在時刻から 7 日

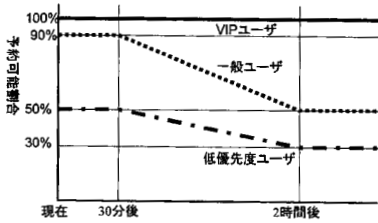


図 6 各ユーザクラスの予約可能割合

```

UtilCheckPeriod = relTime("7d");
MaxReserveCount = 100;
MaxReserveDuration = relTime("2d");
MaxReserveHourNode = 1000.0;

now = absTime(time());
util = plus_rsvutil(
    PLUS_RSV_OWNER,
    now - UtilCheckPeriod,
    now + UtilCheckPeriod);

PLUS_NODE_RESERVABLE =
    ((util[0] <= MaxReserveCount) &&
    (util[1] <= MaxReserveDuration) &&
    (util[2] <= MaxReserveHourNode));

```

図 7 ユーザの予約状況に応じた判断の例

前から 7 日後までの当該ユーザの予約状況を参照し、それが設定した上限値を越えない場合にのみ予約を許している。ここで、`plus_rsvutil` は、予約状況を参照するために追加した組込み関数である。この関数は、引数に、ユーザ名と時間区間をとり、その間のユーザの予約状況をサマライズし、予約回数、予約時間の総計、予約ノード数と予約時間の積の総計を配列として返す。

5.4 実行中のジョブの実行を妨げないポリシー

PBS Professional などでは、実行中の非予約ジョブがある場合には、そのジョブの終了時刻までは、そのノードに対する予約を受理しない。図 8 に示す例は、このようなポリシーを実現した例である。このポリシーは、候補ノードで実行中のジョブを参照し、その終了時刻を `walltime` から推測して、その終了時刻が、予約開始時刻よりもあとである場合には予約リクエストを拒否する。

ここで用いられている `evaluateList` は、われわれが新たに追加定義した `map` 高階関数に相当する組込み関数である。 `evaluateList` は、第 1 引数にリストを、第 2 引数と第 3 引数に変数名をとり、第 1 引数のリストの内容をひとつずつ第 2 引数で指定された変数名にバインドして、第 3 引数で指定された変数を評価し、その結果で構成されるリストを返す。つまり第 2 引数を仮引数として、第 3 引数を関数として評価するのである。この例では、リスト `PLUS_CANDIDATE_NODE.jobs` の内容を `job` にセットし、それぞれに対して `jobFinishTime` を計算している。

```

defaultWalltime = relTime("1:00:00");
now = absTime(time());

jobFinishTime =
    (strcmp(job.state, "Running")==0) ?
    ((job.wallTime isnt undefined) ?
    job.startTime + job.wallTime :
    job.startTime + defaultWalltime) :
    now;

nodeAvailTime =
    (size(PLUS_CANDIDATE_NODE.jobs) > 0) ?
    reverseSort(
        evaluateList(PLUS_CANDIDATE_NODE.jobs,
            "job",
            "jobFinishTime"))[0] :
    now;

PLUS_NODE_RESERVABLE =
    (nodeAvailTime <= PLUS_RSV_START);

```

図 8 実行中のジョブの実行を妨げないポリシー

6. 評価

ポリシーの評価によるオーバーヘッドが予約機構のレスポンスタイムに与える影響を知るために、ポリシー `ClassAd` の評価の時間を計測した。ポリシーがノード情報を参照する場合には、ポリシーの評価時間がシステムに存在するノードの数、およびリクエストされたノードの数に依存することが予想されるため、これらの値を、1,10,100,1000 と変化させて計測した。

測定には `PluS` そのものを用いず、`ClassAd` の評価のみを行うモジュールを作成しそこで行った。これは、`PluS` 内部の他の部分に起因する変動から評価のオーバーヘッドを切り分けるためである。ノードの情報は、ダミーの固定情報となっている。

測定環境には、Athlon 64 X2 2.0G、メモリ 4G、OS Fedora Core 6, Java 処理系には Sun J2SE 1.5.0 を用いた。計測は数百-数万回繰り返し、その平均値を算出している。繰り返し回数は、評価にかかる時間に応じて、それぞれ 1 秒程度経過するように調整している。

図 9, 図 10 にそれぞれ、リスト図 4, 図 5 に示した `ClassAd` の評価にかかった時間を示す。X,Y,Z 軸すべて対数になっていることに注意されたい。

`ClassAd` の評価時間は、短い場合ではマイクロ秒のオーダーであり、もっとも時間がかかっている場合でも 100 ミリ秒に満たないことがわかる。この速度は予約インターフェイスの通信やデータベース更新、キュー制御の時間に比較すると無視できる範囲である。

グラフの傾向としては、図 9 は、要求ノード数に比例した時間がかかっているのに対して、図 10 では全ノード数に対して比例している。これは、前者の評価が必ず成功するため、要求ノード回数だけ評価を行った時点で評価が終了するのに対し、後者は必ず失敗するため要求ノード数によらず、全ノード数だけ評価を繰り返すためである。

このグラフからはわかりにくいですが、図 9 の全ノード

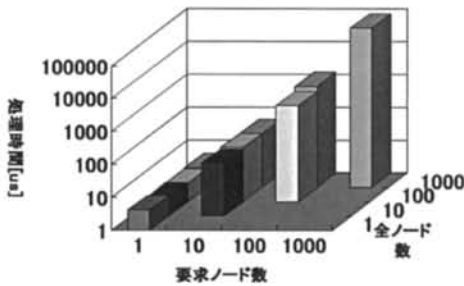


図9 図4に示した ClassAd の評価時間

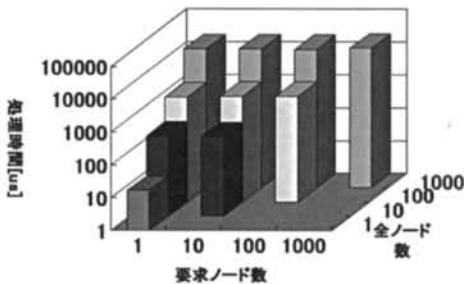


図10 図5に示した ClassAd の評価時間

数が1000である場合の、要求ノード数に対する評価時間は、1, 10, 100, 1000に対して、3.9, 43.0, 1167.0, 72897.8 [μs]と線形を逸脱して増加している。この原因は現在調査中であるが、なんらかのメモリマネージメントの問題である可能性があると考えている。

7. おわりに

事前予約リクエストを受理ポリシーを管理者に記述させる枠組として、ClassAdを用いたポリシー評価機構を事前予約機構PluSに組込んだ。いくつかの管理ポリシーを示し、ClassAdの表現力とPluSの提供する情報とによって、有効なポリシー記述が可能であることを示した。さらに、管理ポリシーの評価にかかる時間を測定し、管理ポリシーの評価によるオーバーヘッドが許容できる範囲であることを示した。

今後の課題は以下の通りである。

- 実環境での運用を通して、実際に管理者が求めるポリシーが現在提示している情報とClassAdの枠組で記述可能であることを検証する必要がある。
- スケジューラが持つ情報をどこまで、どのように管理ポリシーに対して開示するかを検討する。開示する事自体は容易であるが、詳細な情報を開示しても管理ポリシーで解釈する手間がかかったり、ポリシー記述が煩雑になってしまっは意味がない。

ある程度抽象化した情報を提示する必要がある。

- 今回示したシステムで記述できる管理ポリシーは、リクエスト受理のポリシーのみである。今後は、例えば予約開始時刻に割り当てるノードの選択など、より詳細なポリシーを記述できるようにシステムを拡張することを検討している。

謝 辞

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バス網提供方式の開発」による。

参 考 文 献

- 1) Yoshimoto, K., Kovatch, P. and Andrews, P.: Co-scheduling with User-Settable Reservations, *Job Scheduling Strategies for Parallel Processing* (Feitelson, D. G., Frachtenberg, E., Rudolph, L. and Schwiegelshohn, U.(eds.)), Springer Verlag, pp. 146-156 (2005).
- 2) Maui Cluster Scheduler.
<http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php>.
- 3) 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: グローバルスケジューリングのための計算資源予約管理機構, HPCS 2007 予稿集, pp. 127-134 (2007).
- 4) Nakada, H., Takefusa, A., Ookubo, K., Kudo, T., Tanaka, Y. and Sekiguchi, S.: An Advance Reservation-Based Computation Resource Manager for Global Scheduling, *Third Workshop on Grid Computing and Applications* (to appear).
- 5) Condor. <http://www.cs.wisc.edu/condor/>.
- 6) Livny, M., Basney, J., Raman, R. and Tannenbaum, T.: Mechanisms for High Throughput Computing, *SPEEDUP Journal*, Vol. 11, No. 1 (1997).
- 7) Raman, R., Livny, M. and Solomon, M.: Matchmaking: Distributed Resource Management for High Throughput Computing, *Proc. of HPDC-7* (1998).
- 8) Solomon, M.: The ClassAd Language Reference Manual.
<http://www.cs.wisc.edu/condor/classad/>.
- 9) TORQUE Resource Manager.
<http://www.clusterresources.com/pages/products/torque-resource-manager.php>.
- 10) Grid Engine. <http://gridengine.sunsource.net>.
- 11) gLite: Lightweight Middleware for Grid Computing. <http://glite.web.cern.ch/glite/>.
- 12) EGEE Middleware Architecture and Planning, Technical Report DJRA1.4, EU Deliverables.