2003－HPC－95　(22)
2003／8／5

PCクラスタ上での並列分枝限定法の高速化手法

中村心至[†]　山田真太郎[†]
二方克昌[†]　合田憲人[†,††]

最適化問題解法の一つである分枝限定法は，与えられた問題の解を効率よく探索する手法として知られているが，大規模問題の求解には莫大な計算時間を必要とするため，並列化による実行時間短縮が切望されている．本稿では，マスタ・ワーカ方式を用いた並列分枝限定法を MPI，Ninf，AMWAT の 3 つの実装方法を用いて実装した場合の性能評価について述べる．本性能評価の結果，MPI による実装が最も高速な結果を示したが，MPI よりプログラミングが容易な Ninf による実装でも，実装方法の改良により MPI に近い性能が得られることが確認された．

Performance Improvement of a Parallel Branch and Bound algorithm on a PC cluster

Motoyuki Nakamura,[†] Shintaro Yamada,[†] Yoshiaki Futakata[†]
and Kento Aida[†,††]

A branch and bound algorithm, which is one of methods to solve optimization problems, is known as an efficient scheme to search solutions of giver problems. However, performance improvrment of the algorithm by parallelization is needed, because of huge amount of time is necessary to solve problems. This paper evaluates a parallel branch and bound algorithm with master-worker paradigm implemented by three methods, MPI, Ninf, and AMWAT. The performance evaluation results showed that MPI was the fastest, however that Ninf, which has easier programming interface than MPI, also showed close performance to MPI by improvement of an implementation method.

1. はじめに

最適化問題解法の一つである分枝限定法は，与えられた問題の解を効率よく探索する手法として知られているが，大規模問題の求解には莫大な計算時間を必要とするため，並列化による計算時間短縮が切望されている．これに対してマスタ・ワーカ方式を用いて効率よく分枝限定法を並列化する手法が提案され，PC クラスタやグリッドテストベッド上でその性能評価が行われている [1,2,3,4]．マスタ・ワーカ方式は，マスタが複数のワーカを管理することで並列分散処理を実現する方式であり，並列計算システム上での並列プログラミングモデルとして数多く使用されている．[1,2,3,5,8,9]

マスタワーカ方式を用いた並列分枝限定法では，マスタからワーカへのタスクの割り当て方法や，マスタおよびワーカ間での暫定値の更新方法といった要因により，その性能が大きく影響を受ける．そのため，効率のよいタスク割り当てや暫定値更新を実現するアルゴリズムを提案し，実装することが重要となる．その実装方法として，AMWAT[5]，MPI[6]，Ninf[7] が挙げられるが，どのプログラミングツールを使用するかによって，実装されたアプリケーションプログラム（アルゴリズム）の性能や，実装コストが影響を受ける．例えば，マスタ・ワーカ方式を用いたアプリケーションプログラムを実装するためのテンプレートを提供する AMWAT では，ユーザがマスタおよびワーカ上の処理や両者の通信に関する記述をテンプレート中に記述することにより，マスタ・ワーカ方式を用いたアプリケーションを容易に実装することが可能であるが，詳細な通信アルゴリズムを記述することは困難である．それに対して，MPI では，プロセス間の詳細な通信アルゴリズムの記述が可能であるが，AMWAT に比べてプログラムが複雑になるという問題がある．また，Ninf では，MPI よりマスタ・ワーカ方式のプログラミングは容易であるが，AMWAT より複雑であり，通信アルゴリズムの記述においては，AMWAT

† 東京工業大学
Tokyo Institute of Technology
†† 科学技術振興事業団，さきがけ研究 21
PRESTO,JST

より詳細に記述することが可能だが，MPI に比べると詳細な記述を行うことが困難である．

本稿では，これらの問題を解決するため，複数の並列プログラミングツール（AMWAT，MPI，Ninf）により実装されたマスタ・ワーカ方式を用いた並列分枝限定法アプリケーションの性能比較を行う．本稿が対象とする並列分枝限定法アプリケーションは，双線形行列の最大固有値を最小化する解を求めることを目的とした数値最適化問題の１つである BMI 固有値問題の求解アプリケーション [1] である．本評価の結果，MPI による実装が最も高速であるが，MPI よりもプログラミングが容易な Ninf による実装でも MPI に近い性能が保たれることが確認された．

以後、2 節では BMI 固有値問題及びマスタ・ワーカ方式を用いた並列分枝限定法について説明し、3 節では，マスタ・ワーカ方式の実装手法として本稿がとりあげた AMWAT，Ninf，MPI を用いたアプリケーションプログラムの実装について説明する．4 節では実装方法の性能評価について，5 節ではまとめと今後の課題について述べる．

2. BMI 固有値問題解法

本章では，BMI 固有値問題について説明するとともに，並列分枝限定法を用いたマスタ・ワーカ方式について述べる．

2.1 BMI 固有値問題

BMI 固有値問題とは以下の式で定義される双線形行列関数 F(x,y)

$$\mathrm{F(x,y)} := F_{00} + \sum_{i=1}^{n_x} x_i F_{i0} + \sum_{j=1}^{n_y} y_j F_{j0} + \sum_{i=1}^{n_x}\sum_{j=1}^{n_y} x_i y_j F_{ij}$$

$$\mathrm{x} = (\mathrm{x}_1, ..., x_n)^T, y = (y_1, ..., y_n)^T$$

の最大固有値を最小化するベクトル変数 x,y を求める問題であり，以下の式により定義される．

$$\Phi(B) := minimize \Lambda(x,y),$$
$$\Lambda(x,y) := \overline{\lambda}\{F(x,y)\}, (x,y) \in B$$

ここで，$\overline{\lambda}\{F(x,y)\}$ は与えられた x, y に対する行列 $F(x,y)$ の最大固有値, B は x, y の探索空間を表す．

BMI 固有値問題は NP 困難な問題で，$\Phi(B)$ の上限と下限を ε 近傍に大域収束させることにより，有限時間内に ε 最適解を求める分枝限定法アルゴリズムが提案されている．[1]

2.2 マスタ・ワーカ方式を用いた並列分枝限定法

本節では，マスタ・ワーカ方式を用いた並列分枝限定法による BMI 固有値問題の並列解法 [1] について概説する．分枝限定法では，親問題の解の探索範囲を分割して子問題を生成し，各子問題ごとに目的関数の下界値，上界値，解を計算することを繰り返すことにより，最適解を探索する．[1] の手法では，マスタが，解の探索範囲を表す探索ツリーを管理し，以下の手順に従って探索ツリー上の子問題を複数のワーカに割り当てることにより，並列処理を実現する．

(1) **マスタの処理**
マスタは，探索ツリーから下界値が最小である子問題を 1 つ選択し，選択した子問題をワーカに割り当てる．また，マスタは，ワーカ上での計算が終了して返された子問題に対して限定操作を行い，残りの子問題を未処理の子問題のリストに追加する．

(2) **ワーカの処理**
ワーカは，割り当てられた子問題に対し，分枝操作を行って複数の子問題を生成し，生成された子問題の上界値と下界値および解を計算する．次に各ワーカ上の子問題の上界値を比較し，最小の上界値を暫定値として保存する．ここで，下界値が暫定値よりも大きい子問題が発見された場合は，該当する子問題の探索を終了する（限定操作）．

ここでワーカ上での計算粒度に関しては，トレードオフが存在する．並列分枝限定法の計算では，マスタとワーカ間の通信に起因するオーバーヘッド，およびワーカ間の暫定値の更新頻度が全体の計算時間に影響を与える．前者のオーバーヘッドについては，これを減少させることによりプログラムの並列計算効率を向上させることができる．一方，後者のワーカ間での暫定値更新については，これを頻繁に行うことにより各ワーカ上に最新の暫定値が通知されるため，ワーカ上での限定操作を促進し，プログラム全体の計算量を減少させることができる．これら両者はともに計算時間短縮に有効であるが，通信オーバーヘッドを減少させるためには，ワーカ上での計算粒度を増大させることが必要であるのに対し，ワーカ間の暫定値更新頻度を増加させるためには，ワーカ上での計算粒度を縮小させる必要がある．

3. 実 装 方 法

本章では，MPI，AMWAT，Ninf により実装される 3 つの並列分枝限定法アルゴリズムの詳細について述べるとともに，実装方法の違いに起因する 3 つのアルゴリズム間の違いについて述べる．

3.1 MPI による実装

本節では，MPI により実装した BMI 固有値問題解法について述べる．MPI による実装では，rank = 0 のプロセスがマスタ，その他のプロセスがワーカの処理を行うことによりマスタ・ワーカ方式を実現している．

詳細なアルゴリズムを以下に示す．

[**マスタの処理**]

(1) マスタは，まず BMI 固有値問題用パラメータである初期データを読み込む．ここで，初期データとは係数行列 F の次元数とその要素，決定変数 x,y の次元数，分枝の深さなどを意味する．

(2) マスタは，親問題の上界値と下界値を計算し，終了条件（計算した上界値と下界値の相対誤差が指定した値より小さい場合）を満たしているかを判定し，満たしていなければ子問題を生成して，ワーカへ子問題とその初期データおよび暫定値を割り当てる．

(3) マスタは，ワーカからの通信待ち状態となり，ワーカから暫定値更新が通知された場合，その値に基づき限定操作が行われる．

(4) 一方，マスタはワーカから計算終了が通知された場合，子問題計算結果を回収し，終了判定を行う．

(5) 終了していない場合，マスタは残り子問題から最小下界値を選択し，ワーカへ割り当てる．ここで，ワーカに送信されるデータは子問題および暫定値のみで，初期データはワーカ上で保持されている．

(6) (3) に戻り，終了条件を満たすまで繰り返す．

[**ワーカの処理**]

(1) ワーカは，割り当てられた子問題に対し，分枝操作を行って複数の子問題を生成し，生成された子問題の上界値と下界値および解を計算する．

(2) 次に，各ワーカ上の子問題の上界値を比較し，最小の上界値を暫定値として保存する．ここで，暫定値が更新された場合，直ちに暫定解および暫定値が，ワーカからマスタへ通知される．

(3) その後，ワーカ上で限定操作が行われ，終了条件を満たしているかを判定する．満たしていなければ，ワーカは残りの子問題と暫定値をマスタへ返し，待機状態となる．

3.2 AMWAT による実装

AMWAT とは AppLeS(Application-Level Scheduling on the Computational Grid) グループによって作成された，マスタ・ワーカ方式の並列プログラムを容易に開発するためのライブラリである．ユーザは用意された関数テンプレートを用いることでマスタ・ワーカ方式を用いて並列プログラミングを容易に実現できる．

AMWAT により実装されたアプリケーションでは，プログラムの処理がワークサイクルと呼ばれる単位に分けられ，各ワークサイクルごとにマスタ・ワーカ方式でアプリケーション開発者に指定された数のタスク（本稿が対象とするアプリケーションでは，子問題（群）に相当）が実行される．各ワークサイクルでは，全てのタスク実行が終了すると，マスタと全ワーカ間で同期がとられるとともに，プログラムの終了判定が行われ，終了判定を満足しない場合は，次のワークサイクルの処理が開始される．そのため，事前にワーカ上でのタスク実行時間が正確に予想できるようなアプリケーションについては，ワークサイクル内で適切なワーカ間の負荷分散が可能であり，効率のよいプログラムの実行が可能である．しかし，本稿が対象とするアプリケーションでの AMWAT による実装では，マスタとワーカの処理が 3.1 節と以下の点で異なり，MPI による実装に比べるとプログラムの実行性能が低下する可能性がある．

- ワーカ上でのタスク実行時間が静的に決まらないため，各ワークサイクル内での負荷分散が適切に行われない，即ち，ワークサイクル終了時の同期のためにワーカ上で無駄な待ち時間が生じる．
- ワーカ上で新たに最良の暫定値が計算された場合でも，本暫定値の他のワーカへの通知が次のワークサイクル開始時まで行われないため，ワーカ間で暫定値更新効率が低下する．

3.3 Ninf による実装

Ninf は，クライアントマシンからサーバマシン上の計算ルーチン (Ninf executable) の実行を依頼する RPC を実現するためのライブラリである．ここで，クライアントから Ninf executable の呼び出し処理は NinfCall と呼ばれる API を通して行われる．

本稿が対象とするアプリケーションの Ninf による実装 [1] では，Ninf のクライアントプログラムがマスタ，サーバプログラムがワーカに相当しており，マスタからワーカへのタスク（子問題）割り当ては，NinfCall より実現される．ここで，マスタとワーカの処理は基本的に 3.1 節と同様であるが，以下の点で違いがあり，プログラムの実行性能が低下する可能性がある．

- ワーカ上で新たに最良の暫定値が計算された場合

でも，原則として本暫定値のマスタへの通知がタスクの実行終了まで行われないため，ワーカ間で暫定値更新効率が低下する．

- 原則として，タスク（子問題）の割り当て毎にタスクの計算に必要な全てのデータをマスタからワーカへ転送する必要があるため，通信オーバーヘッドが増大する．

本稿では，この 2 つの問題点を改善するため，新たに以下の 2 つの実装を行った．

callback による暫定値更新

本稿では，Ninf の callback 機能 [7] を用い，ワーカ上での暫定値更新が行われた際，直ちに callback が行われ，ワーカからマスタへ暫定値が返されるというアルゴリズムを実装した．具体的には，マスタ上のプログラム内に暫定値更新判定を行う callback 関数を記述し，ワーカ上で暫定値更新が行われると，ワーカが callback 関数を呼び出し，暫定値をマスタへ通知する．マスタでは，ワーカから通知された暫定値をもとにマスタ上の暫定値を更新する．

ワーカ上でのデータ保持

本稿が対象とする BMI 固有値問題初期データは静的なデータであるため，一度マスタからワーカに転送された場合，それをワーカ上で保持できることが望ましい．これによって，データ転送量減少による通信オーバーヘッドの削減が可能である．本稿では，ワーカ上で最初に転送された初期データ保持を可能にする実装を行った．まず，初期データ，子問題，暫定値をマスタからワーカに，具体的には，各ワーカへの初回タスク割り当て時に限り，転送する．各ワーカプロセスは Ninf_call_executable_t を用いて実装されているため，初回起動後プロセスがマシン上で維持されている．このプロセス上で初期データを別途に保存しておき，2 回目以降の呼び出しでは保存しておいたデータを計算に使用することにより，初期データを送信しないようにした．

4. 性 能 評 価

本節では，3 節で述べた実装方法の性能比較を行うための性能評価について述べる．

4.1 性能評価環境

表 1 に性能評価を行った PC クラスタの性能を示す．表 1 の PC クラスタ上の 1 ノード上でマスタプロセスを実行し，他のノード上でワーカプロセスを実行した．ここで，MPI による実装では，SCore 上の MPI を使用した．各ソフトウエアのバージョンは，SCore は SCore 4.2.1，MPICH は MPICH − 1.2.4，AMWAT は AMWAT Ver. 1.0，Ninf は Ninf Ver. 1 である．

また，性能評価に用いたベンチマーク問題として，実問題であるヘリコプターの旋回動作制御問題，及び性能評価のために変数 x,y 及び行列 F の大きさや値を人工的に生成した擬似問題を用意し，それぞれの解を計算した．

4.2 AMWAT，MPI，Ninf の性能比較

表 2 に PC クラスタ A 上での AMWAT，MPI，Ninf の性能比較結果を示す．表中の Depth はワーカ上での計算粒度を表しており，具体的には，Depth=1 とは 1 回の分枝操作によって 1 つの問題を 2 個の子問題に分割することを示しており，Depth=3 とは 3 回分枝操作を繰り返し，1 つの問題を最終的に $\sum_{i=1}^{3} 2^i = 14$ 個の子問題を生成することを示している．

表 2 より，最も高速であるのは MPI，その次に Ninf，最後に AMWAT であることがわかった．まず，MPI と Ninf を比較すると，Ninf はタスク（子問題）割り当て毎に初期データを含めた全てのデータをマスタからワーカへ転送するため，通信オーバーヘッドが増加するということ，ワーカ上で暫定値更新が行われても現在計算中の処理を終了しなければマスタに暫定値が通知されないため，暫定値更新の遅延が起こるという問題があるので Ninf は MPI より遅くなっている．また，MPI と AMWAT を比較すると，ワークサイクル終了時の同期のためにワーカがアイドリングしてしまうこと，マスタ上で暫定値が更新されても次のワークサイクル開始時まで反映されないため，暫定値更新の遅延が起こるという問題があるので AMWAT は MPI より遅くなっている．さらに，Ninf と AMWAT を比較した場合，AMWAT はワークサイクル終了時の同期が必要なため，ワーカのアイドリング時間が長くなっている．これによって，AMWAT は Ninf より遅くなっている．

4.3 MPI，Ninf の性能比較

表 3 に，PC クラスタ B 上でのヘリコプターの旋回動作制御問題，擬似問題 $n = 5$ および $n = 6(n_x = n_y = n)$ の 3 つの問題を MPI および，$Ninf$ による実装で計算した場合の計算時間を示す．

表 3 より，全ての問題に対し，MPI による実装が最も高速であることがわかる．3 節でも述べたように，MPI による実装では，AMWAT や Ninf と比較して，高速なアルゴリズムの記述が可能である．また，MPI が Ninf と比べて高速であるのは，NinfCall によるオーバーヘッドが MPI のメッセージ通信に比べて大きいということが考えられる．

表 1　実験環境

PC クラスタ A	node	(PentiumIII Xeon 550MHz $\times 4 CPU$, $Memory 1GB$) $\times$ $2 nodes$ (PentiumIII Xeon 700MHz $\times 4 CPU$, $Memory 1GB$) $\times$ $2 nodes$
	network	100Mbps ehternet
	OS	Linux 2.4.19
PC クラスタ B	node	(PentiumIII 1.4GHz $\times 2 CPU$, $Memory 512 MB$) $\times$ $18 nodes$
	network	100Mbps ehternet
	OS	Linux 2.4.2

表 2　AMWAT，MPI，Ninf の性能比較（Depth=1））

問題	計算時間 [sec]			
	ワーカ数	AMWAT	Ninf	MPI
ヘリコプター制御問題	1	4659	3273	3131
	2	2473	1634	1562
	4	1278	695	658
	8	692	361	329
	16	—	261	167
擬似問題 (n=5)	1	3391	2947	2779
	2	1738	1470	1395
	4	881	622	591
	8	472	324	308
	16	—	188	166
擬似問題 (n=6)	1	31385	29282	27708
	2	15912	14854	13844
	4	8152	6015	5627
	8	4255	3063	2868
	16	—	1654	1488

表 3　MPI，Ninf の性能比較（Depth=1）

問題	計算時間 [sec]				
	ワーカ数	Ninf	callback	データ保持	MPI
ヘリコプター制御問題	1	1238	1238	1226	1159
	8	158	158	156	145
	16	86	87	85	73
	32	77	75	76	37
擬似問題 (n=5)	1	1156	1153	1134	1110
	8	152	152	148	145
	16	82	82	79	78
	32	51	52	50	45
擬似問題 (n=6)	1	13232	13251	12882	12647
	8	1603	1605	1525	1516
	16	817	816	786	770
	32	453	452	420	402

4.3.1　Ninf とデータ保持の性能比較

表 3 より，Ninf については，特にデータ保持を実装することで，従来の Ninf の実装と比べて計算時間短縮が実現されていることが確認された．１つの問題を解く際に，表 3 の実験ではマスタからワーカへの NinfCall が多く行われ，特に擬似問題（n=6）では NinfCall が 23513 回行われており，初期データ転送量を削減した有効性が表れていると言える．また，同様の擬似問題（n=6）では問題サイズが大きめで，転送データ量を減少させ，通信オーバーヘッドを減少させるため，データ保持がより有効的であるということがわかる．Ninf の実装の中で最も速かったデータ保持の実装における実行時間は，従来の Ninf での実装における実行時間より，約 1.08 倍の速度向上率があった．

4.3.2　Ninf と callback の性能比較

次に，従来の Ninf での実装と callback の実装について比較する．表 6 に従来の Ninf による実装と callback を用いた実装による計算時間およびマスタ上での暫定値更新回数を示す．

表 6 より，callback を実装した Ninf と従来の Ninf での実装による結果を比較するとそれほど callback の有効性が見られなかった．特に，Depth が小さい時は，従来の Ninf と callback の暫定値更新回数を比較して，それほど変わらないということがわかった．これは，計算粒度 (Depth) が小さい時ほど，ワーカ上でのタスク処理が高速で終了し，callback を行わなくても，暫定値更新が頻繁に行われるため，本来，暫定値更新頻度の増加を促進する目的である callback の有効性は薄いためと考えられる．また，計算粒度（Depth）が大きくなるほど，暫定値更新回数に差が出ていることから，計算粒度（Depth）が大きい場合には，滞る暫定値更新を促進させることで計算時間短縮が可能であると思われる．しかし，今回の実験では，計算粒度が最小の場合の性能が最も高く，あまり有効でないということが確認された．

4.3.3　SCore 上でのトランキングの性能評価

MPI による実装に関しては，SCore システム上でのネットワークトランキングを利用した実行時間も測定したが，ほぼ，ネットワークトランキングしていない場合の実行時間と差が出なかった．性能に差が出なかった原因として，ヘリコプターの旋回動作制御問題と擬似問題 $n = 5$ は問題サイズが小さく，また，2 つの問題サイズより大きい擬似問題 $n = 6$ の場合においても，1 回のデータ転送の際に，118KB のデータが転送されているが，ネットワークがボトルネックになるほどではなかったと考えられる．よりデータ転送量

表 4 Ninf と callback の計算時間および暫定値更新回数の比較

問題	計算時間 [sec](暫定値更新回数 [回])			
	ワーカ数	Depth	Ninf	callback
ヘリコプター制御問題	1	1	1238(33)	1238(35)
		2	1369(27)	1370(33)
		3	1510(29)	1517(48)
		4	1868(29)	1873(50)
		5	1969(31)	1972(56)
	32	1	77(38)	75(38)
		2	54(30)	53(34)
		3	50(28)	49(46)
		4	67(30)	67(50)
		5	67(29)	66(62)
擬似問題 (n=5)	1	1	1156(63)	1153(69)
		2	1212(45)	1220(63)
		3	1292(41)	1300(75)
		4	1405(36)	1407(75)
		5	1444(32)	1452(75)
	32	1	51(66)	52(74)
		2	53(52)	53(72)
		3	59(45)	58(68)
		4	72(46)	69(89)
		5	79(33)	78(98)
擬似問題 (n=6)	1	1	13232(64)	13251(66)
		2	13734(50)	13825(69)
		3	14420(49)	14303(76)
		4	15941(45)	16482(71)
		5	16430(35)	16527(75)
	32	1	453(73)	452(76)
		2	462(52)	465(74)
		3	482(51)	485(75)
		4	547(44)	544(77)
		5	601(36)	582(74)

の多い大規模問題においては，ネットワークトランキングが有効であると思われる．

5. ま と め

数値最適化問題の 1 つである BMI 固有値問題をマスタ・ワーカ方式を用いた並列分枝限定法により求解するアプリケーションを MPI，AMWAT，Ninf の 3 つの方法で実現し，性能比較を行った．性能評価の結果，MPI による実装が最も高速であることが確認されたが，MPI よりプログラミングが容易な Ninf についても実装手法の改良によって MPI に近い性能が得られることが確認された．

今後の課題として，1 節で述べたように，マスタ・ワーカ方式を用いた並列分枝限定法では，マスタからワーカへのタスクの割り当て方法，マスタおよびワーカ間での暫定値更新方法により，性能が大きく影響する．よって，並列分枝限定法を高速化するために，最適なタスク割り当て，暫定値更新を考慮した手法を提案していく予定である．

参 考 文 献

[1] K. Aida, Y. Futakata, and S. Hara. High-performance parallel and distributed computing for the bmi eigenvalue problem. In Proc. The 16th IEEE International Parallel and Distributed Processing Symposium, 2002.

[2] J. Goux, S. Kulkarni, J. Linderoth, and M. Yoder. An enabling framework for master-worker applications on the computational grid. In Proc. the 9th IEEE Symposium on High Performance Distributed Computing (HPDC9).

[3] M. O. Neary and P. Cappello. Advanced Eager Scheduling for Java-Based Adaptively Parallel Computing. In Proc. of the 2002 joint ACM-ISCOPE conference on Java Grande, 2002.

[4] Y. Tanaka, M. Sato, M. Hirano, H. Nakada, and S. Sekiguchi. Performance evaluation of a firewallcompliant globus-based wide-area cluster system. In Proc. of 9th IEEE Symposium on High-Performance Distributed Computing, 2000.

[5] AMWAT: AppLeS Master-Worker Application Template http://apples.ucsd.edu/amwat/

[6] MPI: The Message Passing Interface (MPI) standard http://www-unix.mcs.anl.gov/mpi/

[7] Ninf: A Global Computing Infrastructure http://ninf.apgrid.org/

[8] E.Heymann, M. A. Senar, E. Luque, and M. Livny. Adaptive scheduling for mastar-worker applications on the computational grid. In Proc. of the 1st IEEE/ACM International Workershop on Grid Computing(Grid2000), 2000.

[9] G. Shao, F. Beerman, and R. Wolski, Master/slave computiong on the grid. In Proc. Heterogeneous Computing Workshop, 2002.