2004－HPC－99　(6)
2004／7／30

階層的マスタワーカ方式を用いたグリッドアプリケーションにおける負荷分散の性能評価

大角知孝[†]　合田憲人[†,††]

本稿では，複数の PC クラスタから構成されるグリッド上で階層的マスタワーカ方式によるグリッドアプリケーションにおいて，各 PC クラスタ間のタスクの移動による負荷分散アルゴリズムの性能評価について述べる．本性能評価の結果，WAN 上の計算機間の通信遅延にかかわらず PC クラスタ間で積極的にタスクを移動させることが性能向上につながることが確認された．

Performance Evaluation of Load Balancing Algorithms for the Grid Application Parallelized with the Hierarchical Master-Worker Paradigm

Tomotaka Osumi[†] and Kento Aida[†,††]

This paper evaluates load balancing algrithms, which moves tasks among PC clusters, for the parallelized with the hierarchcal master-worker paradigm. The performance evaluation results showed that the algorithm to aggresively move tasks among PC clusters is most effective under WAN settings with various communication latencies.

1. はじめに

グリッド計算は，ネットワークに接続された計算資源を安全に安定して，且つ安易に利用することで高性能計算を行うこと技術であり，低コストで大規模計算資源を利用することができ，科学技術計算を始め，高性能計算を必要とする分野でその利用が期待されている．グリッド計算の計算モデルのひとつとして階層的マスタワーカ方式[2] が提案されている．階層的マスタワーカ方式では図 1 に示されるようにスーパバイザ，マスタ，ワーカの 3 つの階層的な役割を各計算機に割り当て，スーパバイザが複数のマスタへタスク割り当てを行い，さらにマスタはそのマスタが統制するワーカへタスク割り当てを行い計算処理をする．階層的マスタワーカ方式では，マスタ間の負荷分散を適切に行うことが性能向上のために重要であるが, 負荷分散アルゴリズム[3] についてはまだ十分に研究されていない．

本稿では，階層的マスタワーカ方式によって並列化されたグリッドアプリケーションにおける負荷分散アルゴリズムの性能評価について報告する．本稿で対象とする階層的マスタワーカ方式では，グリッド上の複数の PC クラスタそれぞれにマスタと複数のワーカが配置され，スーパバイザが各マスタが保持するタスク数を逐次監視し，状況に応じてマスタ間のタスクの収集と分配を行うことで負荷分散を実現する．本性能評価の結果，WAN 上の通信遅延によらず積極的に PC クラスタ内でタスクを移動させることが性能向上につながることがわかった．

以後,2 節では本研究が扱う計算モデルとその並列化手法である階層的マスターワーカの説明し，3 節ではアプリケーション上に実装した負荷分散アルゴリズムを説明する．4 節ではその評価結果について，5 節ではまとめと今後の課題について述べる．

2. 計算モデル

本稿が対象とする計算モデルである階層的マスタワーカ方式では，図 1 に示すようにマスタとそのマスタが統制するワーカ群によるグループが複数構成され，各グループのマスタをスーパバイザが統括している．マスタと複数のワーカから構成されるグループ内では，通常多くの通信が発生するためこれらを同一サイト内の計算資源，例えば PC クラスタ上に配置することにより，通信を局所化でき，アプリケーションの

† 東京工業大学
Tokyo Institute of Technology
†† 科学技術振興機構さきがけ
PRESTO,JST

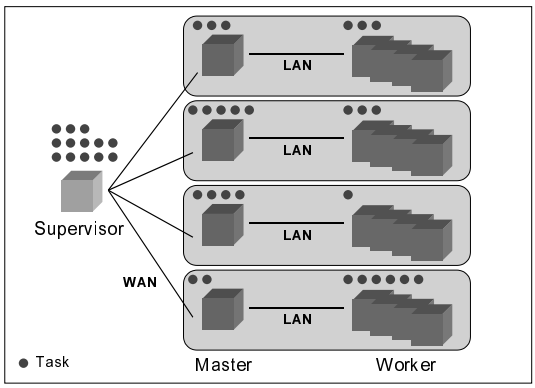

図 1 階層的マスタワーカ方式

実行性能を向上させることができる．従って，スーパバイザマスタ間の通信は WAN 経由で行われることを想定しており，マスタワーカ間は LAN 経由で行われることを想定している．

本方式におけるアプリケーションの実行では，マスタとワーカからなるグループそれぞれが，与えられた問題の部分問題を計算し， その結果をスーパバイザが管理することで問題全体を計算する．具体的には，スーパバイザはまず部分問題の計算をタスクとしてマスタに割りあてる．マスタは，割り当てられた部分問題をさらに分割してタスクを生成し，複数のタスクをそれぞれ複数のワーカに割り当て，計算結果を受け取る．また， 計算状況に応じて，各グループ間で負荷の不均衡が生じるため，スーパバイザがマスタの保持しているタスク数を監視し，各マスタからタスクを収集，再分配することで負荷分散を実現する．

階層的マスタワーカ方式を用いて並列化されたアプリケーションの例として，並列分枝限定法のアプリケーション[2] が挙げられる．分枝限定法は最適化問題の実用的な解法として広く用いられている手法であり，与えられた問題 (親問題) の解の空間を分割し子問題を生成し，各子問題ごとに目的関数の下界値，上界値，解の計算を繰り返すことにより最適解を探索する．これらの過程は，与えられた問題を根ノード，分枝操作により生成された子問題を葉ノードとする探索木により表現される．探索木上の子問題はそれぞれ独立に計算できるため，異なるワーカ上で子問題をタスクとした並列計算が可能である．以下，階層的マスタワーカ方式を用いて並列化された並列分枝限定法アプリケーションにおけるスーパバイザ，マスタ，ワーカ上の処理の詳細を解説する．

2.1 ワ ー カ

ワーカは以下のように自分を統括するマスタから子問題とその時点でのマスタの保持していた最小の上界値 (暫定値) を受け取り，計算を行って結果を同じマスタに返す．

(1) マスタから割り当てられた子問題に分枝操作を行い，新たな子問題を生成する．
(2) 生成した子問題について上界値と下界値を計算し，暫定値と比較して算出した上界値が小さい場合は新しい暫定値として更新する．
(3) 暫定値を上回る下界値をもつ子問題を計算対象 (探索木) から除外する，即ち限定操作を行う．
(4) 限定操作の結果残った子問題と暫定値をマスタに返す．

2.2 マ ス タ

マスタは，以下に示すように，ワーカへの子問題割り当て及び計算結果の回収と，スーパバイザからの処理要求 (リクエスト) への対応という 2 つの処理を行う．以下の処理はスーパバイザから計算終了のリクエストを受けるまで続けられる．

スーパバイザからのリクエスト対応

マスタはスーパバイザからのリクエストの有無を定期的に確認し，リクエストの内容に応じて以下の処理を行う．

(1) リクエストが途中計算結果回収の場合，マスタがその時点で保持している未処理の子問題の数，暫定値とその暫定解，そのマスタ上に保存されている最小下界値をスーパバイザへ通知する．
(2) リクエストが暫定値更新である場合，スーパバイザから送信された暫定値が現在のマスタ上に保存される暫定値よりも小さい場合，マスタの暫定値をスーパバイザから送信された暫定値に更新する．
(3) リクエストが子問題の要求である場合，指定された数の子問題をスーパバイザに転送する．
(4) リクエストが子問題割り当てである場合，新たに割り当てられた子問題を受け取り，マスタ上の未処理子問題のリストに加える．

ワーカへの子問題の割り当て

マスタは各ワーカの状態を調査し，アイドル状態にあるワーカに以下の処理を行う．

(1) アイドル状態のワーカが計算結果を保持している場合は，ワーカから子問題および暫定値等の計算結果を受け取る．
(2) ワーカから受け取った暫定値がマスタ上に保存されている暫定値よりも小さい場合はマスタ上の暫定値をワーカから受け取った暫定値に更新する．またワーカから受け取った子問題に限定操作

を行う．

(3) マスタ上の未処理子問題をワーカに割り当てるとともに，マスタ上に保存されている最新の暫定値を同ワーカへ通知する．

2.3 スーパバイザ

スーパバイザは定期的にマスタと通信を行い，マスタ間の負荷分散を担い，終了条件を満たすまで以下を繰り返す．

(1) 各マスタに途中計算結果回収リクエストを送信する．

(2) マスタから受け取った暫定値がスーパバイザ上に保存されている暫定値よりも小さい場合はスーパバイザ上の暫定値をマスタが受け取った暫定値に更新する．また，各マスタに暫定値更新のリクエストを送信する．

(3) スーパバイザ上に保存されている暫定値と最小下界値から計算の終了条件を判定し，ユーザが指定した終了条件を満たす場合は計算を終了する．

(4) 3 節で説明するアルゴリズムに従い，負荷の高いマスタに子問題要求のリクエストを送信し，子問題を受け取る．また負荷の低いマスタに子問題割り当てのリクエストを送信し，子問題を転送する．

3. 負 荷 分 散

階層的マスタワーカ方式では，マスタの負荷分散を適切に行うことが，アプリケーションの性能向上のために重要である．負荷分散アルゴリズムは，スーパバイザがマスタ間の子問題の移動を開始する条件である発生条件と，子問題の移動を行う際の移動方法により決定される．

3.1 発 生 条 件

スーパバイザは定期的に各マスタ上の未処理子問題に関する情報を収集し，マスタ間で子問題を移動させることにより，マスタ間の負荷分散を実現する．本稿では子問題の移動の発生条件として以下の 3 つの方法について議論する．

(1) i 番目のマスタの未処理子問題数 $qsize_i$ が以下の式を満たすことを発生条件とする．

$$qsize_i = 0 \quad (for\ some\ i)$$

即ち本アルゴリズムでは，任意のマスタ上で未処理子問題がなくなった時点で子問題の移動を開始する．

(2) $qsize_i$ が以下の式を満たすことを発生条件とする．

$$qsize_i < desNodes_i \times \alpha \quad (for\ some\ i)$$

ここで $desNode_i$，α はそれぞれマスタ i 上の理想未処理子問題数，定数 $(0 < \alpha < 1)$ を意味する．即ち本アルゴリズムでは，任意のマスタ上の未処理子問題数が理想保持子問題数の α 倍を下回った時点で子問題の移動を開始する．$desNodes_i$ は，i 番目のグループ (マスタとマスタ総括するワーカ) の他のプロセスにより負荷も考慮した実質的な計算能力を数値化したものを $mCap_i$ とすると，

$$desNodes_i = \sum_j qsize_j \times mCap_i/(\sum_j mCap_j)$$

で定義され， 計算能力により重み付けして全未処理子問題を分配した場合に各マスタが保持する未処理子問題数を意味する．尚，$mCap_i$ はスーパバイザがマスタの情報収集する度に逐次更新される．

(3) スーパバイザの定期的なマスタに関する情報収集毎を行う毎に，子問題の移動を開始する．

3.2 子問題の移動方法

スーパバイザは 3.1 節で述べた発生条件が満足された場合に子問題の移動を行う．本稿では，子問題の移動方法として以下の 2 つについて議論する．

(A) スーパバイザはマスタ i に対して以下の式の $Nodes_i$ で算出される数の子問題を回収または割り当てる．

$$Nodes_i = qsize_i - \sum_j qsize_j \times mCap_i/(\sum_j mCap_j)$$

ここで．$Nodes_i$ が正であればスーパバイザはそのマスタから子問題を回収し，負であれば分配する．本アルゴリズムでは，子問題の移動により各マスタの子問題数を理想保持子問題数にすることを意味する．

(B) スーパバイザはマスタ i に対して以下の式の $Nodes_i$ で算出される数の子問題を回収または割り当てる．

$$Nodes_i = qsize_i - \sum_j qsize_j \times nWorkers_i/(\sum_j nWorkers_j)$$

ここで $nWorkers_i$ はマスタ i が統括するワーカ数を意味する．本アルゴリズムでは計算能力を考慮せずマスタが統制するワーカ数に応じて均等分配するように子問題を移動する．

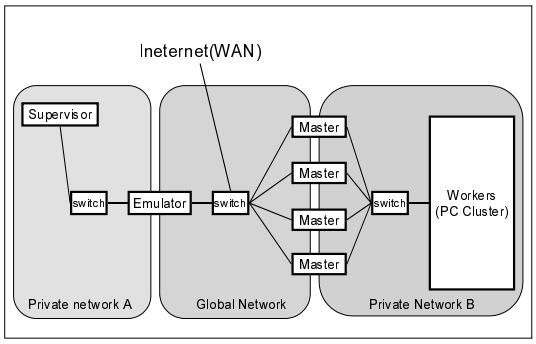

図 2 擬似グリッド実験環境のネットワーク構築図

4. 性能評価

本節では，3 節で述べた負荷分散アルゴリズムの性能評価について述べる．

4.1 実験環境と性能評価問題

本評価では，図 2 および表 1 に示す擬似グリッド実験環境上で負荷分散アルゴリズムの評価を行った．実際の WAN 上の計算資源により構成されるグリッド環境では，ネットワークの混雑，バンド幅などは時々刻々と変化するため，再現性のある実験結果を得られないという問題点がある．しかし，本擬似グリッド実験環境では，図 2 のようスーパバイザとマスタの間に設置されたルータ PC 上でネットワークエミュレータソフトウェア NISTnet[7] を実行することにより WAN 上の通信遅延を考慮したスーパバイザと各マスタ間通信を再現できる．

ワーカに相当する計算ノードは同一の PC クラスタ内の 32CPU から構成されるが，これらを仮想的に 4 分割し，8CPU から構成される 4 つの PC クラスタが接続されている環境を擬似的に再現する．

スーパバイザとマスタの通信は WAN 経由になるため，セキュリティを考慮した処理が必要となる．本擬似グリッド実験環境では，スーパバイザとマスタに相当する計算ノードには Globus2.4[5] がインストールされており，スーパバイザからマスタへの処理要求は GridRPC ミドルウェアである Ninf-G 1.1.1[6] を用いて実現される．一方，マスタとワーカ間の通信は LAN または PC クラスタ内ネットワークを経由するため，マスタからワーカへのタスク割り当てはより高速な RPC 機能を提供する Ninf[6] により実現される．

ベンチマーク問題は BMI 固有値問題の並列分枝限定法による解法を取り上げた．BMI 固有値問題は，以下の双線形行列関数 $F(x,y)$，

$$F(x,y) = F_{00} + \sum_{i=1}^{n_x} x_i F_{i0} + \sum_{j=1}^{n_y} y_j F_{0j} + \sum_{i=1}^{n_x}\sum_{j=1}^{n_y} x_i y_i F_{ij}$$

$$F_{ij} = F_{ij}^T (i = 0, ..., n_x, j = 0, ..., n_y)$$

$$x = (x_1, ..., x_{n_x})^T, y = (y_1, ..., y_{n_y})^T$$

の最大固有値を最小化するようなベクトル変数 x，y を求める問題であり，最適解を求める分枝限定法アルゴリズム[4] が提案されている．

性能評価に用いる BMI 固有値問題は，一様乱数によって $F(x,y)$ 中の係数行列 F_{ij} を生成することにより，9 個のベンチマーク問題 (RD05-01〜05，および RD06-01〜04) を用意した．これらの問題の問題サイズは，RD05-01〜05 では変数 x，y の次元数，行列 F_{ij} の行列サイズ (m) は $n_x = n_y = 5, m = 20$ であり，RD06-01〜04 では $n_x = n_y = 6, m = 24$ である．

4.2 発生条件に関する性能評価

本節では 3.1 節で述べた負荷分散発生条件に関する性能評価の結果について述べる．発生条件に用いる 3.1 節のアルゴリズム (2) については，α の値を 0.2 としたもの (2a) と 0.5 としたもの (2b) の 2 つの場合について評価を行った．子問題の移動方法は 3.2 節のアルゴリズム (A) を採用した．

スーパバイザとマスタ間の通信は，往路復路それぞれに片道 0，15，25，50，100msec の遅延を発生させた条件下で評価を行った．これを実際のインターネット環境で考えると，東京工業大学すずかけ台キャンパス (神奈川県横浜市) から大まかに 15msec が九州大学 (12.35msec)，25msec が沖縄大学 (26.09msec)，50msec がシンガポールのネットワーク団体である SingAREN(76.23msec)，100msec がマサチューセッツ工科大学 (94.15msec) に相当する．

以上の環境下で 9 個のベンチマーク問題を用いて評価を行ったところ，実行結果はほぼ全ての問題に同様の傾向が見られたため，ここでは，RD06-02 の実行結果のみを提示し考察する．

図 3 はアプリケーションの計算開始から終了までの実行時間を示す．全てのアルゴリズムについて図 ref-fig:ratio より通信遅延が大きくなるほど実行時間も増加していることがわかる．アルゴリズム毎の性能を比較すると全ての通信遅延の条件下において (3) が最も高性能であり，(2b)，(1)，(2a) という順で性能が低くなっている．

通信遅延による実行時間増加の原因は 2 つ考えられ

表 1　擬似グリッド実験環境の計算機構成

	CPU	Memory/ノード	OS	CPU 数
Worker(PC cluster)	Pentium3(1.4GHz)Dual	512MB	Linux	32
Master(PC)	Pentium4(2.4GHz)	512MB	Linux	4
Supervisor(PC)	Pentium4(2.4GHz)	1GB	Linux	1
Network Emulator(PC)	Pentium4(2.4GHz)	1GB	Linux	1

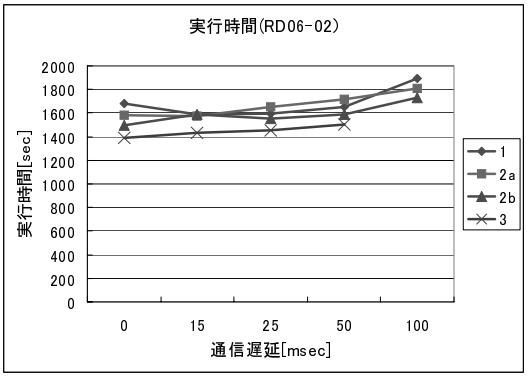

図 3　実行時間 (RD06-02)

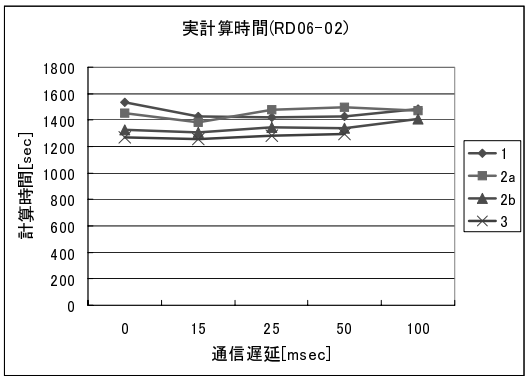

図 4　実計算時間 (RD06-02)

る．1 つはグリッドアプリケーションを実行するためにスーパバイザがマスタ上に Ninf-G のプロセスを立ち上げるための初期化処理と計算終了後にプロセスを終了させる終了処理があり，これらの処理は各マスタに逐次実行されるため通信遅延が増えるに従い時間が大幅に増加する．例えば RD06-02 の計算時には通信遅延が 0msec のときは初期化と終了処理をあせて 120 秒程度だが，100msec になると 340 秒程度となる．2 つめの原因，マスタ間の子問題の移動にかかる時間が遅延により大きくなることである．図 4 は初期化と終了処理を除いた実際に子問題の処理をしている時間 (実計算時間) に示す．図 4 より，アルゴリズムを比較すると実計算時間に差はあるものの通信遅延による実計算時間への影響には決まった傾向が見られるわけではない．即ち，通信遅延が大きくなることによる実行時間の増加は主に初期化と終了処理による影響が大きく，子問題の移動の遅延による影響は少ないことがわかる．

次に，図 5 はマスタが処理する問題がなくなりアイドル状態にあるマスタのアイドル時間について 4 つのマスタの平均を取った結果を示す．図 5 より (3)，(2b) については通信遅延の増加によるアイドル時間の増加は非常に小さいが，(2a)，(1) では非常に大きいことがわかる．アルゴリズム毎に比較すると，アイドル時間は毎回負荷分散を発生させる (3) が最も小さく，計算ノードがマスタになくなった時点で負荷分散させる (1) が最も大きい．

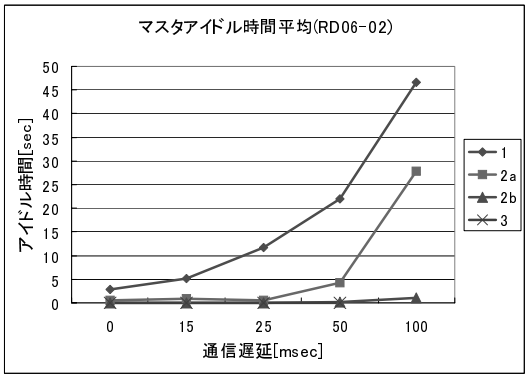

図 5　マスタアイドル時間平均 (RD06-02)

4.3　子問題の移動方法に関する性能評価

本節では 3.2 節で述べた 2 つの子問題の移動方法の違いによる性能評価の結果について述べる．測定での子問題の移動方法は 3.2 節の (A) と (B) の 2 つであり，その発生条件は 4.2 節の結果より最も高速であった 3.1 節の (3) を採用した．また本評価では，グリッド上での PC クラスタの性能が他のジョブ (Job) の影響により異なる場合の性能を評価するために 1 つの PC クラスタ (マスタ，ワーカのグループ) 上で NAS Parallel Benchmark[8] の LU(クラス C) を実行し負荷を与えながら BMI 固有値問題のベンチマーク問題 RD06-01〜04 を実行した．

図 2 はその実計算時間を，図 6 はワーカへ割り当てられたタスク数 (子問題数) の平均を示す．図 6 で

表 2 ベンチマーク問題の実計算時間

問題	実計算時間 [sec]			
	アルゴリズム (A)		アルゴリズム (B)	
	他 Job 無	他 Job 有	他 Job 無	他 Job 有
RD06-01	445	576	454	567
RD06-02	1210	1683	1208	1572
RD06-03	713	943	753	905
RD06-04	998	1344	1004	1284

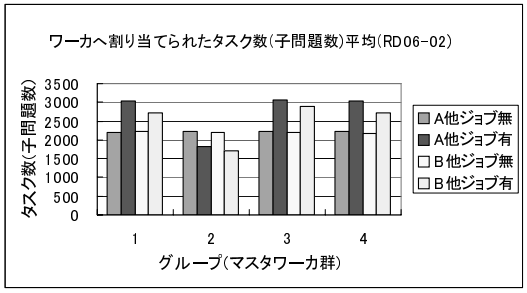

図 6 ワーカへ割り当てられたタスク数平均 (RD06-02)

は，各マスタは 8 つのワーカに対してそれぞれ子問題の割り当てを行うため，各ワーカに割り当てられた子問題の平均を示している．また，負荷を加えたのはグループ 2 の 8 つのワーカである．

図表 2 より実計算時間は他のジョブの有無にかかわらず 2 つのアルゴリズム間に大きな差は見られないことがわかる．またワーカへ割り当てられた子問題数も，他のジョブによって負荷を加えた場合，グループ 2 だけその負荷のために減少しているが 2 つのアルゴリズム間には大きな差が見られる．本評価では用いた発生条件が 3.1 節の (3) である．スーパバイザがマスタの情報を収集するたびに子問題の移動を行うため，この場合どちらの子問題の移動方法を用いても各マスタがアイドル状態になることが無かったため，実計算時間に影響を及ぼさなかったと思われる．

5. おわりに

本稿では，複数の PC クラスタからなるグリッド上で階層的マスタワーカ方式により並列化されたグリッドアプリケーションにおける負荷分散アルゴリズムの性能評価結果を報告した．擬似グリッド実験環境上に階層的マスタワーカ方式を用い並列化した並列分枝限定法アプリケーションを実行し評価したところ，負荷分散アルゴリズムについて以下の点が明らかになった．発生条件については，スーパバイザ・マスタ間の通信遅延によらず子問題をマスタ間で積極的に移動させることが性能向上につながると考えられる．子問題の移動方法については，スーパバイザが頻繁に子問題の移動させる条件下ではマスタが保持する未処理子問題に不足や偏りが生じにくく，子問題の移動方法による性能への影響は見られなかった．

グリッド上での実用性を考慮した場合，3.1 節の (3) のようにスーパバイザが頻繁に子問題移動を行うことは，WAN 上の通信トラフィックを不必要に増大させるため，問題が発生することも考えられる．よって，今後は通信トラフィックと負荷分散発生回数のトレードオフも考えた性能評価を行う予定である．

参 考 文 献

1) 合田 憲人，二方 克昌，原 辰次，”並列分散計算システム上での BMI 固有値 問題解法”，情報処理学会論文誌ハイパフォーマンスコンピューティングシステム，42(SIG12(HPS4))，pp.132-141，2001.
2) Kento Aida, Wataru Natsume, Yoshiaki Futakata, ”Distributed Computing with Hierarchical Master-worker Paradigm for Parallel Branch and Bound Algorithm” , Proc. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003), 2003
3) Rob V. van Nieuwpoort, Thilo Kielmann, Henri E. Bal, ”Efficient load balancing for wide-area divide-and-conquerapplications” , Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming (PPoPP'01), 2001
4) H. Fujioka and K. Hoshijima. ”Bounds for bmi eigenvalue problem”. Trans. of the Society of Instrument and Control Engineers, 33(7):616-621, 1997.
5) Globus: http://www.globus.org/
6) ninf: A Global Computing Infrastructure. http://ninf.apgrid.org/
7) NIST net: http://www.itl.nist.gov/div892/itg/carson/nistnet/
8) NAS Parallel Benchmark: http://www.nas.nasa.gov/NAS/NPB/