耐故障/耐高負荷を考慮した並列分枝限定法と基本性能の評価

久保田 和人[†] 仲瀬 明彦[†]

マスタ/ワーカモデルを用いた並列分枝限定法に対して，ワーカジョブに優先度を与え優先度に応じてジョブを多重実行することで，耐故障/耐高負荷性能を付加した．PC クラスタを用いたシミュレーション実験により，高負荷なワーカの処理がボトルネックとなり全体の処理時間が増大する問題に対して，本手法が有効に機能する場面を確認した．また，あらかじめ想定した台数のワーカのダウンに対して，多重度を適切に設定することにより処理の停止を回避できることを確認した．

Parallel Branch and Bound Method with Fault and Load Tolerance and Its Performance Evaluation

Kazuto Kubota[†] and Akihiko Nakase[†]

A load and fault tolerant feature is embedded in a master-worker style parallel branch and bound algorithm. In our method, a priority is assigned to each worker job, and only high priority jobs are executed redundantly. Experimental results on a PC cluster show its load and fault tolerant feature. When heavy load workers exist, the execution time of the basic branch and bound algorithm increases. But when our algorithm is used, the execution time is stable. The basic algorithm cannot complete its work, when some workers hang up during execution. But our algorithm can complete its job, in case that the number of stopped workers is under the given threshold value.

1. はじめに

分枝限定法は，最適化問題を解く手法として広く使われている．計算機クラスタやグリッド[1] のような計算機環境の登場にともなって，より大規模な問題が高速に解けるようになり，活用の場面はますます広がりを見せつつある．しかし，計算機規模の大規模化や広域分散化はシステム全体の安定性に対して不利に働く．特定の計算機ノードが故障すると並列計算が終了しなくなる可能性があり，また，特定のノードのレスポンスが何らかの理由で遅延すると，その処理がボトルネックとなって全体の処理時間が著しく伸びる可能性がある．レスポンスが遅延する要因としては，他のユーザのジョブによる計算機の負荷の上昇や，ネットワークの混雑などが考えられる．以下では，これらの要因を高負荷という言葉で代表させることにする．また，故障と高負荷を合わせて障害と呼ぶことにする．このような障害の起こった状況下でも安定して動作する分枝限定法を提供できればアプリケーションユーザの受けられる恩恵は多大なものとなる．

計算に耐障害性を持たせる方法は大きく分けて 3 つある．(1) 物理的に冗長性を持たせる方法[2] は，ハードウェア自体を多重化させる方法であり，1 つのハードウェアがダウンしても他のハードウェアが処理を継続することで計算のダウンを防ぐ．(2) ミドルウェアあるいは OS に耐障害性を持たせる方法[3] は，故障によって消失した処理を何らかのソフトウェア手段によって復旧する方法である．たとえば，処理のスナップショットをとり，正常なハードウェア上で処理を再開する．(3) アプリケーションそのものに冗長性を持たせる方法[4),5] は，OS やミドルウェアに頼らずにアプリケーションだけで障害に備えるというものである．なお，これら 3 つの手法は排他な手法ではなく，お互いに組み合わせることで耐障害性を高めることができる．

本稿では，アプリケーションレベルで分枝限定法に耐障害性を持たせる方法について述べる．並列分枝限定法はマスタ/ワーカ型モデルを仮定し，ワーカ処理を多重化することで障害に備える．しかし，すべてのワーカ処理を多重化すると処理数が膨大になるので，処理に優先度を付け優先度の高いもののみ多重度を上

† 株式会社東芝研究開発センター
Research & Development Center, TOSHIBA Corporation

げて実行するというアプローチをとる．多重化によるメリットは，(1) 故障でジョブが消失しても処理全体が停止しないことや，(2) 多重化したジョブのうちの最も早く返ってきた結果を採用することで処理全体の遅延を防げる可能性があることなどである．本稿では処理の詳細を述べた後に，基本的な動作の把握，耐高負荷，耐故障性能の評価を行う．

2. 分枝限定法と最良優先探索

分枝限定法は最適化問題を解く解法の 1 つである．解空間全体を木で表し，ルートから葉までのパスが 1 つの解と見なされる．たとえば，最適化問題の 1 つであるナップザック問題は，重さ w と価値 v を持つ n 個の荷物が存在したときに，重さの合計が W 以下で価値の総和が最大となる荷物の選び方を求める問題である．分枝限定法を用いてナップザック問題を解く場合は，木の深さ d の節点を d 番目の荷物を入れるか入れないかに対応させる．木全体では，すべての荷物を入れるか入れないかというすべての場合が尽くされていることになる．この中で価値が最も高くなる荷物の入れ方，すなわち，その入れ方を示すルートから葉へのパスが求めるナップザック問題の解となる．

解の探索はルートから葉方向へ木を生成しながら行う．節点を分岐させて新たな節点を作る操作を分枝操作と呼ぶ．これは元の問題を小さな部分問題に分けることに相当する．個々の節点において，その節点から下に木を広げていった場合，到達の可能性がある最大の評価値を計算することができる．この値を上界値と呼ぶ．また，個々の節点において，少なくとも達成可能である評価値を計算できる．この値を下界値と呼ぶ．ある節点の上界値が他の節点の下界値を下回ったとき，その節点からの探索では最適解に到達できないことが分かる．したがって，その節点からの分枝操作を中止する．この処理を限定操作と呼ぶ．分枝限定法では，無用な解の探索を打ち切ることで解の探索空間を限定している．

一般に木を探索する手順としては，深さ優先探索や幅優先探索などがある．これらの探索では，分枝操作をあらかじめ決められた順序で行う．一方，これとは別に個々の節点に対して探索の優先度を求め優先度の高い順に節点の処理を行う方法がある．この方法は最良優先探索と呼ばれる．これは，解が存在する可能性が高い方向の探索を優先して行うことで，早く最適解に到達することを目指すものである．優先度を決める方法として，各節点における下界値を優先度として用いる方法がある（最良下界探索と呼ばれる）．これは，

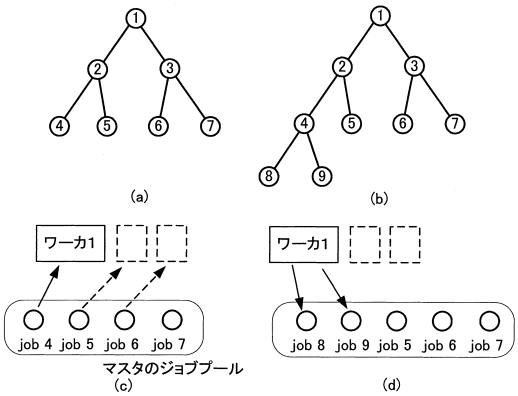


図 1 並列分枝限定法におけるジョブの割当て
Fig. 1 Parallel branch and bound method.

下界値の大きい節点は小さい節点より最適解に到達する可能性が高いであろうという考え方に基づくものである．並列分枝限定法への耐障害機能の付加にあたっては，冗長に実行されるワーカジョブの数を限定するために，この優先度を利用する．

3. 並列分枝限定法への耐障害機能の付加

マスタ/ワーカ型の並列分枝限定法では，マスタがジョブプールを管理しジョブプール内には子問題（節点）が格納される．マスタは 1 つ以上の子問題をジョブとしてまとめワーカに投げる．ワーカはジョブ内の子問題に対して分枝操作を行い，新たに発生した子問題および下界値をマスタに返す．この様子を**図 1** に示す．図 1 (a) の節点 4〜7 がこれから解かれようとしている子問題であり，個々の子問題が 1 つのジョブであるとする．マスタのジョブプールに格納された job 4 は，空きワーカ 1 に投げられる（図 1 (c)）．ワーカ 1 は分枝操作を行う．簡単のため，ワーカは 1 回のみの分枝操作を行うものとする．ワーカ 1 は 2 つの節点を生成しマスタへと返す．マスタは 2 つの節点 8，9 をジョブプールへと格納する（図 1 (d)）．このときの全体の木の様子は図 1 (b) のようになる．

耐障害機能を付加するにあたって，各節点に探索の優先度を付ける．ここでは単純に下界値を優先度とする．図 1 (a) の木に優先度をつけた木を**図 2** (a) に示す．マスタ中のジョブ管理は，図 2 (c) に示すような拡張されたジョブプールと多重度リストで行う．拡張されたジョブプール（以降，単にジョブプールと呼ぶ）は，個々のジョブについてジョブ番号，優先度，実行度を保持し，優先度の順番にソートされる．実行度にはそのジョブを実行しているワーカの台数が保持される．多重度リストにはジョブプール中のジョブの順位ごと

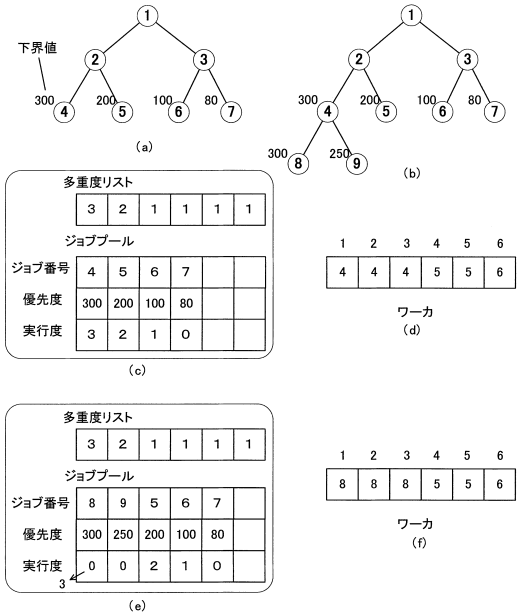


図 2 並列分枝限定法への耐障害機能の付加
Fig. 2 Fault and load tolerant parallel branch and bound method.

の多重度の上限が格納される．同じジョブでもジョブプール内の順位が変動することで多重度が変化することに注意されたい．ジョブの実行は，実行度が多重度に満たないジョブを空きワーカに投げることで行う．ジョブプールからのジョブ削除は，該当するジョブを実行するワーカがマスタに結果を返した時点で行う．ジョブプール中のジョブがなくなった時点で処理の終了となる．

図 2 (c) は図 2 (a) の 4〜7 のノードを実行しているときのジョブプールの様子である．多重度リストには，優先度の高い順に 3，2，1，1，... が与えられているものとする．ジョブ 4 が 3 台，ジョブ 5 が 2 台，ジョブ 6 が 1 台のワーカでそれぞれ実行されている．これらのジョブを実行しているワーカの様子を図 2 (d) に示す．図 2 (d) において，ワーカ 1 が処理を終了しマスタに新たなジョブ 8，9 を返したとする．それぞれの下界値を 300，250 とする．マスタ内のジョブプールからジョブ 4 が削除され，新たにジョブ 8，9 が挿入される（図 2 (e)）．ここでジョブ 8 を 3 重に実行したとするとジョブ 8 の実行度は 0 から 3 となり，このときのワーカの様子は図 2 (f) のようになる．また，このときの木の様子は図 2 (b) のようになる．

耐障害機能の付加は，並列分枝限定法に 2 つの利益を生む可能性がある．1 つ目は，高負荷なワーカが発生しても全体の計算性能を著しく低下させない効果である．高負荷なワーカが発生し，そのワーカに優先度の高いジョブが割り当てられてしまうと，有効であろう探索が停滞してしまい下界値の更新が行われず早い段階での枝刈りが行われなくなる可能性がある．耐障害機能を付加することで，有効な方向の探索は多重化され，停止，停滞しにくくなるので計算性能の著しい低下を回避できる可能性がある．2 つ目は，耐故障効果である．優先度が低く多重化されていないジョブであっても，他の価値が高いジョブが終了してジョブプール内での順位が上がると多重化されて実行されるようになるので，多重度リストの先頭要素の値未満のワーカが故障しても計算は最後まで行われることになる．

4. 評価プログラムの実装

以下，本稿では耐障害機能を付加した並列分枝限定法の基本性能について評価していく．評価にあたってはナップザック問題を並列に解くコードを利用して様々なケースのシミュレーションを行った．ここでは実装の詳細を述べていく．

先に述べたように，マスタ/ワーカモデルで並列分枝限定法を解く場合，各節点の分枝操作がワーカのジョブとなる．単純に各節点の 1 回の分枝操作を 1 ジョブとする方法もあるが，これでは粒度が細かすぎるため，複数の節点をワーカに配り，各節点を起点とする分枝操作も複数回行うことにする．今回用いたプログラムでは，ワーカが 1 回のジョブで分枝操作を行う上限回数（*branch_limit*）を設け，分枝操作がこの回数を超えた場合はその時点で残っている節点（子問題）と下界値をマスタに返送する．

1 つのジョブに入れる節点数は *unit_number* という変数で管理する．節点はスタックに保持されており，そこから *unit_number* 個だけ切り出してジョブ化する．スタック中の節点が *unit_number* 個に満たない場合は，半分の節点をジョブ化する．これは，空き PE 数が多くマスタの節点が少ない場合，すべての節点を 1 台の PE に投げると残りの PE に節点が渡らないので，残った節点を半分ずつにしながら空きワーカに配るというアプローチをとったためである．

スタック中の節点のジョブ化は，マスタがワーカにジョブを投げるタイミングで行う．まず，先に述べた個数の節点をスタックから切り出してジョブを作成する．このジョブをスタックジョブと呼ぶ．次に，スタックジョブとジョブプール中のジョブの中から評価関数を用いて実行するジョブを選択する．評価関数については後述する．選択されたジョブがスタックジョ

ブだった場合は，スタックジョブをジョブプールに登録する．そうでなかった場合はスタックジョブを破棄し，含まれている節点をスタックに戻す．このような操作を行うことで，ジョブに含まれる節点数がなるべく *unit_number* に近づくようにしている．

ジョブに含まれる情報は，ジョブ番号，優先度，実行度および節点数である．ジョブの優先度は，含まれる節点の下界値の最大値としている．ジョブプール内のジョブは優先度でソートされる．実行するジョブは，実行度が多重度リストの値より小さいものの中から選択される．選択のポリシとしては，(1) 優先度が高いものを選ぶ，(2) 実行度の小さいものを選ぶなどが考えられる．ここでは，スタックジョブの優先度が一番高い場合はスタックジョブを，そうでない場合は，ジョブプール中で優先度が高いジョブを選択している．ジョブはマスタがその時点で持つグローバルな下界値とともにワーカに投げられ，実行度が 1 加算される．なお，今回の実装では多重度リストは実行中に変化させていない．

ワーカからジョブ（節点群）と下界値が返送された場合，節点群はマスタのスタックに登録され下界値が更新される．なお，多重化されているジョブに関しては，最初の 1 つの結果が返ってきた段階で残りのジョブは不要となるが，現段階では，これらのジョブの実行は中止せずに終了するのを待つものとする．将来は，不要となったジョブを kill する機能まで含めた実装に拡張する予定である．

ワーカには，ジョブの処理時間をコントロールする機能を組み込んでいる．具体的には，分枝操作の途中に wait loop を入れ見掛け上の処理時間を遅くする機能を組み込んでいる．これによって，ワーカに負荷が発生している状況を擬似的に作り出すことができる．

処理の全体の流れをまとめると以下のようになる．

マスタプログラム

- Step 1. 多重度リストをユーザが入力（本アルゴリズムでは多重度の更新は行わない）．
- Step 2. スタックに木のルート節点を置く．
- Step 3. ワーカからのジョブの実行結果が届いていたら，ワーカを空きワーカリストに登録．得られた結果が，すでに他のワーカによって実行済みの結果ならば Step 4. へ．そうでなければ以下の処理を実行．
  - Step 3-1. 終了したジョブをプールから削除．
  - Step 3-2. 返された節点をスタックに登録．
  - Step 3-3. 下界値の更新．
- Step 4. スタック中の節点からスタックジョブを生成．ジョブプール中のジョブとスタックジョブの中から実行度が多重度に満たないジョブを選び，その中から評価関数を用いて実行すべき 1 つのジョブを選択する．
- Step 5. 選択したジョブがスタックジョブならばスタックジョブをジョブプールに登録．そうでなければスタックジョブを破棄し，含まれる節点をスタックに戻す．
- Step 6. 選択したジョブを空きワーカで実行．実行度を 1 加算．
- Step 7. ジョブプールが空なら終了．そうでなければ Step 3. へ．

ワーカプログラム

- Step 1. マスタからジョブを受信．
- Step 2. 節点をスタックに登録．
- Step 3. スタック中の節点が空か，ループ回数が *branch_limit* に到達するまで以下の処理を繰り返す．
  - Step 3-1. 節点を取り出し分枝操作を実行．下界値の更新．新たに生成された節点をスタックに戻す．
  - Step 3-2. 擬似的な負荷を与えるための wait loop.
- Step 4. スタック中の節点と下界値をマスタに返送．

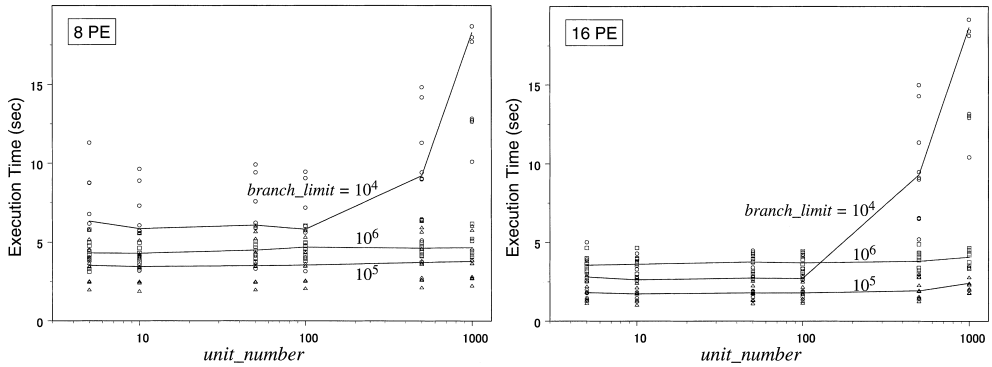
表 1 プログラムのパラメータとその目的
Table 1 Program parameters and their purposes.

unit_number	（粒度制御，負荷分散）
branch_limit	粒度制御，（負荷分散）
多重度リスト	多重度の制御

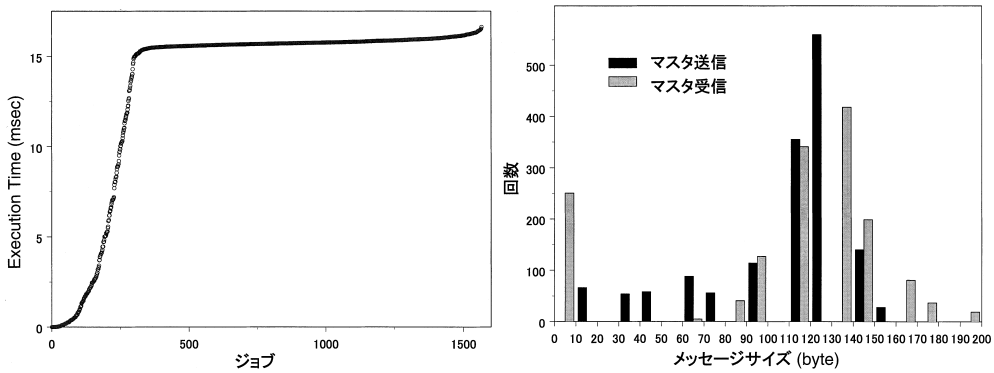
本プログラムで制御できるパラメータと目的を**表 1** に示す．一般に *unit_number* を小さくするとジョブ内の節点の数が減り粒度は下がる．しかし，仮にジョブ内に含まれる節点が 1 つでもそこから生成される部分木は巨大になる可能性があるため，粒度を決めるパラメータとしては間接的といえる．一方，*branch_limit* はワーカ内での分枝操作に上限を与えるので，粒度を決定するパラメータとしては直接的である．また，マスタワーカモデルでは粒度が下がれば負荷分散は均等化される方向に進む．多重度リストはプログラムの耐障害性をコントロールするパラメータである．

5. 評価実験と考察

前章で説明したプログラムを用いて，本手法の挙動，性能を評価していく．ここで評価する項目は以下の 3 項目である．



(a) unit_number と branch_limit を変化させたときの処理時間



(b) 計算時間と通信データサイズ（16 PE）

図 3 基本性能の測定
Fig. 3 Basic performance.

- 多重度リストを変化させたときのプログラムの挙動
- 高負荷なワーカが存在する場合の性能
- ワーカに故障が発生した場合の耐故障性能

実験は 16 台構成 32 CPU の PC クラスタを用いて行った．諸元を**表 2** に示す．プログラムは MPI（mpich-1.2.5.2）を用いて C 言語で記述した．本環境下でのノード間通信のショートメッセージのレイテンシは片道約 600 μsec，バンド幅は 10.7 Mbyte/sec であった．

以降の実験ではワーカ数を 1 から 31 まで変化させている．15 ワーカまでは，マスタおよびワーカをそれぞれ別々の計算機ノードに割り当てている．16 から 31 ワーカのときは，マスタを実行している計算機ノードには 1 つのワーカを，それ以外の計算機ノードには最大 2 つのワーカを割り当てている．

ナップザック問題は，与えられた入力データによって挙動が大きく異なるため，ここでは 10 種類の問題を人工的に生成し，その平均的な挙動を観察することにした．作成したデータは，荷物の重さの最大格差を 2 倍とし，価値/重さを 1%から 4%まで変動させ，荷物の数を 100 から 250 まで変えたデータの中から，1 台での処理時間が 10 秒以上，60 秒未満のものを 10 個選択したものである．これらは，枝刈りは発生するが探索にはある程度の時間を要する問題である．

表 2 PC クラスタのノードの仕様
Table 2 PC cluster specification.

台数	16 台（32 CPU）
Node CPU	Dual Pentium-III 800 MHz
Node memory	1 GB
Node HDD	60 GB
Node NIC	Intel eepro 100（100 BASE-T）
Switch	Cisco Catalyst 3500

5.1 予備実験

評価に先立って予備実験を行い，以降の実験で用いる *unit_number*（ジョブ中の節点数）と *branch_limit*（ジョブの粒度）を決定した．*unit_number* を 5，10，50，100，500，1000 とし，*branch_limit* を 10^4，10^5，10^6 と変化させたときの処理時間を 8 PE と 16 PE で計測した．結果を**図 3** (a) に示す．横軸に *unit_number*，

表 3 冗長に使う PE 数と多重度リストの作り方
Table 3 Redundant PEs and redundancy list.

冗長 PE 数	多重度リストの構成方法
0	11111...
1	21111...
2	31111..., 22111...
3	41111..., 32111..., 22211...
4	51111..., 42111..., 33111..., 32211..., 22221...

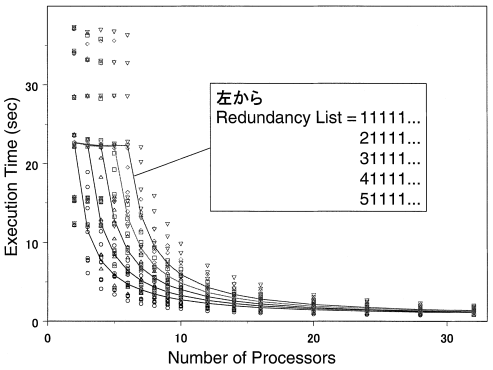
縦軸に処理時間をとり，*branch_limit* ごとにグラフが描いてある．8 PE，16 PE の場合ともに *branch_limit* を 10^5 とした場合に処理時間が短くなること分かる．このとき，*unit_number* が処理時間に与える影響は少ないが，*unit_number* = 10 のときに，わずかながら処理時間が短くなっている様子が観察できたので，以降の実験では *unit_number* = 10，*branch_limit* = 10^5 として実験を行っていくことにする．

図 3 (b) に，16 PE で上記パラメータを用いたときのジョブの計算時間と通信データサイズを示す．計算時間のグラフには，個々のジョブの計算時間が描かれており，ジョブは処理時間の短い順にソートして番号が振られている．大部分のジョブが 15 msec を少し超える程度の処理時間であることが分かる．通信データサイズのグラフは，マスタがワーカに送信したデータサイズおよびマスタがワーカから受信したデータサイズの分布を示している．送受信のデータサイズは，ともに 120 byte 付近を中心に分布していることが分かる．今回利用した環境で，120 byte のメッセージの通信時間を計測したところ，およそ 730 μsec であった．

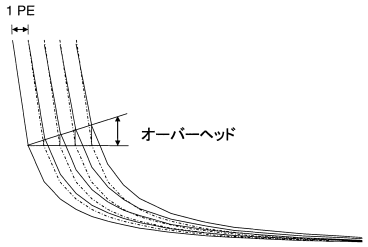
5.2 多重度リストを変化させたときの挙動の把握

冗長に使う PE を 0 台から 4 台まで変化させて処理時間を計測した．冗長な PE 数が同じでも多重化リストの実現方法は複数ある．この様子を**表 3** にまとめる．これらの構成法に関して PE 数と処理時間の関係を測定した．**図 4** (a) は，多重度リストの先頭の値を 1，2，3，4，5 と増やしたときの PE 数と処理時間の関係である．PE 数が小さいときは，多重度リストの先頭の値が大きいほど処理時間を要しており，PE 台数が増えるに従って処理時間は耐故障性のない 1111... のケースへと収束している．

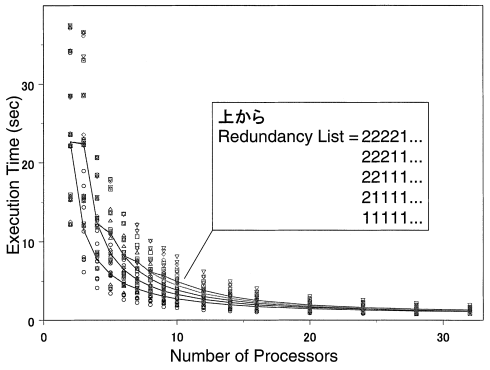
1111... のグラフを右側に冗長 PE 数分だけシフトして図 4 (a) のグラフに重ねたものを図 4 (b) に示す．2111... のケースは，ほぼ重なっている．これは 2111... の場合，2 つのワーカが同じ仕事をするので見かけ上はワーカが 1 台減って 1 ワーカで仕事をしていることになるからである．3111... 以上の場合も同様な考え方ができる．処理のオーバヘッドがない状況だとグ



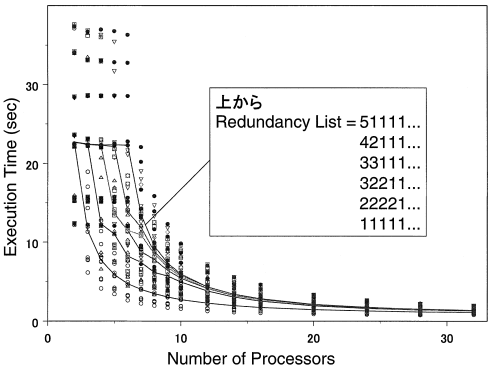
(a) 多重度リストの先頭の値を変化させた場合



(b) 挙動の把握



(c) 多重度リストを先頭から 2 で埋めた場合



(d) 冗長 PE 数 4 の場合

図 4 多重度リストの構成法と処理時間への影響
Fig. 4 Effect of redundancy list for performance.

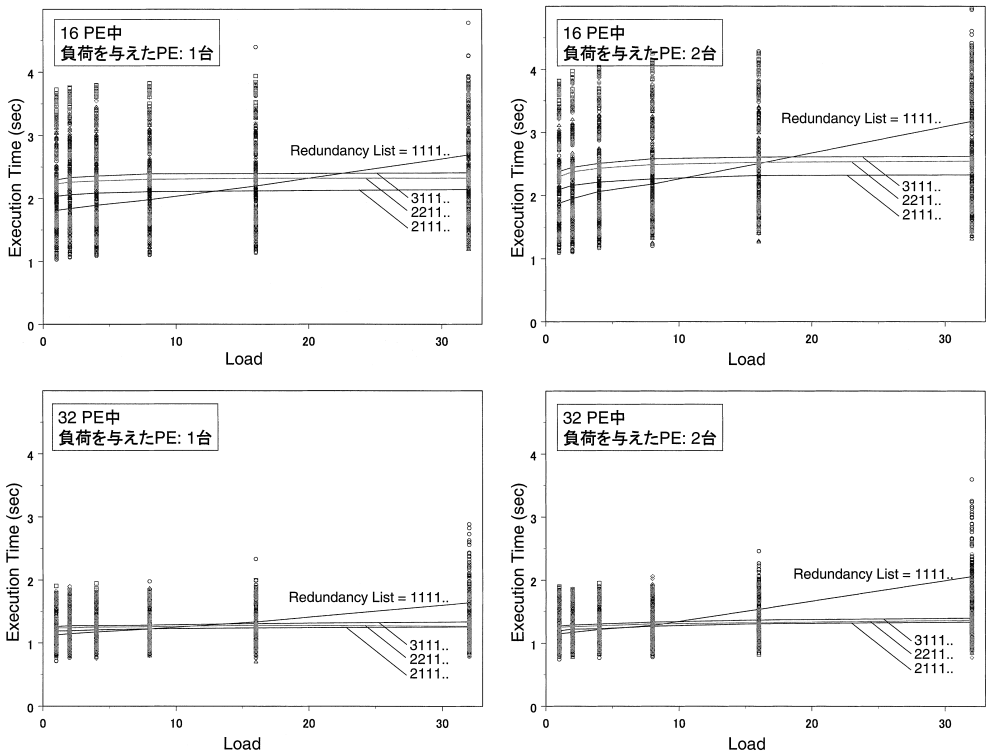


図 5 高負荷 PE 存在時の性能
Fig. 5 Load tolerance performance.

ラフは重なることになるが，実際にはオーバヘッドがあるためにずれが生じる．多重度リストの先頭の値が大きくなるにつれて，ずれ（オーバヘッド）が大きくなっていることが分かる．PE 数が多くなると処理性能に対する 1 台の差は見えなくなるので，すべてのグラフは 1111... のケースへと収束することになる．

図 4 (c) は，多重度リストを 11111...，21111...，22111...，22211...，22221... と変えていった場合の PE 数と処理時間の関係である．同じ冗長 PE 数であるにもかかわらず，図 4 (a) と比べた場合，21111... のグラフに他のグラフが沿うように寝ているのが分かる．これは，冗長 PE 数を 1 増やすことでいきなり実効 PE 数が 1 減ったような挙動を示すのではなく，21111... の状態から実効 PE 数が 1 減った状態にゆるやかに近づいていくことを示している．

冗長 PE 数を 4 にした場合の 5 通りの処理時間を図 4 (d) に示す．すでに述べたように PE 数が少ない 10 程度以下の領域では，リストの先頭を大きくして後ろに 1 を並べる 51111... という構成よりも，多重度を均等化する 22221... という構成の方が高速であることが分かる．これらの中間的構成法である 322...，33...，42... は処理時間も 51111... と 22221... の間となっている．同じ冗長 PE 数にもかかわらず処理時間に差が出るのは，それぞれの場合に同時に実行できるジョブ数が異なるからである．たとえば，合計ワーカ数が 5 の場合，51111... だと結局 1 種類のジョブしか実行しないことになるが，22221... だと 3 種類のジョブを実行することができ，より広い探索を行えることになる．本手法の適用範囲は PE 数が大きな場合を想定しているので，PE 数が小さい部分の振舞いを厳密に調べる必要性は少ないが，多重度リストの構成法と処理時間の関係を理解しておくことは，本手法の動作の理解および使用時に多重度リストの値を決定するうえで重要である．

5.3 高負荷な PE が存在する場合の性能

高負荷で処理が遅延する PE が存在した場合の本手法の性能を調べた．PE 数は 16，32 とし，負荷を与える PE 数は 1，2 とした．負荷は 1（負荷なし）から 32（性能が 1/32 に低下）まで変化させ，2 台に負荷を与える場合は同一の負荷とした．各ケースについて 10 種類の問題を 10 回ずつ計 100 回実行している．このとき，負荷を与える PE はランダムに決定した．多重度リストは 1111...，2111...，3111...，2211... の 4 種類を用いた．**図 5** に負荷を横軸に，処理時間を縦

軸にとったグラフを示す．

耐障害性を考慮しないと（1111... のケース），負荷が増大するに従って処理時間が伸びていくことが分かる．負荷を与える PE 数が 2 の方が，この傾向が顕著である．一方，耐故障性を考慮した手法では処理時間の変化はさほどなく，高負荷な PE が存在しても動作が安定している．耐障害性を考慮しない場合に処理時間が伸びるのは，負荷の高い PE が良い解を生む節点（クリティカルパス）をかかえてしまうと，その間に他の速いノードが無駄な方向の探索を行ってしまうためである．

2 PE に負荷を与えた場合，冗長な PE 数が 2 の場合だけでなく，1 でも性能がほぼ維持できている．これは，クリティカルな計算が 2 台の負荷の高い PE に 2 重化して割り当てられるケースが確率的に少ないためだと考えられる．耐障害性を考慮することにより処理が高速になるケースは，16 PE の場合は負荷が 10〜20 程度以上，32 PE の場合は負荷が 10 程度以上である．32 PE の場合は耐障害性を考慮することによるペナルティがほぼゼロなので，本手法を適用することによる処理時間面での不利益はない．

5.4 耐故障性能

ワーカが障害でダウンする場合を想定して，ワーカ故障時に探索が最後まで終了するか否かを調べる実験を行った．故障ワーカを 1，2，4，8，16 と変化させ，多重度リストの内容は 1111...（耐障害なし），2111...，3111...，5111...，9111...，17111... と変化させた．それぞれの組合せについて先に利用した 10 種類のデータをそれぞれ 10 回ずつ計 100 回実行し，処理が最後まで終了した回数をカウントした．利用 PE 台数は 32 台である．なお，各実験において故障させるワーカはランダムに選択し，故障はワーカが最初のジョブを受け取った瞬間に発生するものとした．

表 4 に結果を示す．表中の○は理論的にプログラムは正しく実行されることが期待され，実際にも 100 回正しく終了した組合せである．3 章で述べた「多重化されていなかったジョブも処理が進んだ段階で多重化されて実行される」という機能が，期待どおりの耐故障動作を行っていることを示している．表中の×は理論的にプログラムは停止し，結果もすべてハングアップしてしまった組合せである．表中の数字が入っている部分は，原理的にプログラムの終了が保証できなくても，実際には処理が正しく終了する可能性のあるケースである．たとえば，32 PE 中の 4 PE が故障する場合を仮定したとき，多重度リストを 3111...（冗長な PE 数が 2）とすれば，100 回中 98 回プログラムの停止を回避できていることが分かる．これは，4 台の故障 PE が存在しても，1 つのジョブに対して故障 PE ばかり 3 台が割り当てられる確率は小さいことを示している．この理論的確率は，全 PE 数，故障 PE 数，多重度リストの値から計算できる．この値はプログラムの動作が確率的に保証できればよい問題に対して有用なパラメータとなる．

表 4 耐故障性能の評価（32 PE 使用）
Table 4 Evaluation of fault tolerance with 32 PEs.

故障 PE 数	1	2	4	8	16
1111...	×	×	×	×	×
2111...	○	99	81	39	10
3111...	○	○	98	85	24
5111...	○	○	○	100	87
9111...	○	○	○	○	100
17111...	○	○	○	○	○

○：理論，計測ともにプログラムが実行完了
×：理論，計測ともにプログラムは実行停止

今回の実験では，多重度リストの値を固定していたためにジョブの多重度に上限が存在し，故障 PE がこの上限を上回った場合にプログラムが停止してしまう状況が生じている．アルゴリズムに実用性を持たせるためには，このような状況を回避するための拡張が必要である．たとえば，実行中のジョブが 1 つだけの場合，多重度リストの値を超えたジョブの多重化を行うことでプログラムの停止を回避することが可能となる．

6. 今後の課題

6.1 他の耐高負荷手法との比較

高負荷である PE の存在が全体の処理時間に与える影響は，ジョブの粒度が大きいほど高く，小さいほど低いと考えられる．これは，負荷の高い PE がクリティカルパスにある節点をかかえた場合，粒度が大きいと結果がマスタに返るまでに処理時間を要してしまい，その間に他の PE が無駄な処理を行ってしまう可能性があるが，粒度が小さければ，クリティカルパスに関わる結果がマスタに返るまでの時間も短くなり，その間に行われている他の PE の無駄な処理も少なくて済むからである．今回の実験で使用した粒度は，高負荷な PE が存在しない場合には，ほぼ適切な値である．しかし，高負荷な PE が存在した場合は，その負荷の大きさに応じて適正な粒度が変化する可能性がある．この負荷と適正粒度の関係，および，このような状況下での本手法の有効性について検証していきたい．

グリッド上でのアプリケーション実行を支援するミドルウェアである Ninf-G2[6] は，関数（ジョブ）にタイムアウト機能を導入している．これは，タイムアウ

トしたジョブを他の PE で実行させることで耐高負荷性を実現するものである．この方法は，処理時間が読めないジョブに適切なタイムアウトを設定することが難しいという問題と，タイムアウトしてからジョブの実行を開始するので，その間の時間が無駄になるという問題点がある．しかし，耐障害性のために PE を消費しないので，利用可能な PE をすべて計算に利用できるというメリットがある．このようなタイムアウトを利用した方法と本手法との定量的比較も行っていきたい．

6.2 不要ジョブ，過剰に多重化されたジョブの処理

ジョブが多重化されていた場合，最初の答えが返ってきた時点で多重化されて実行された残りのジョブは不要になる．このとき，積極的に不要ジョブを kill すれば，空きワーカを増やすことができる．また，新しいジョブがジョブプールに投入された時点で，多重化して実行されていたジョブの優先度の順位が下がり，実行度が多重度リストの値を下回るケースが生じる．これは，ジョブが過剰に多重化されていると見なすことができ，過剰分を kill することで先のケースと同様に空きワーカを増やすことができる．空きワーカを増やすことは探索の範囲を広げ高速化に結び付く反面，ワーカを kill する処理はオーバヘッドをともなう可能性がある．これらの処理を付加することの有効性について検証していきたい．

6.3 開発環境の整備

既存の分枝限定法のコードに本手法を適用する場合，ジョブプール管理に関するコードの追加もしくは変更が必要となる．この作業はさほど煩雑ではないが，この処理がミドルウェア化，API 化されていればユーザのプログラム開発コストは大きく削減される．今回評価に使ったコードから分枝限定法本体の部分と耐障害性のために付け加えたジョブプールやワーカの管理部分のコードを切り分け，耐障害性を持つ分枝限定法記述用ミドルウェア，API を構築したいと考えている．

6.4 実用環境での利用，評価に向けて

本手法が有効に活用される場面は，Grid のような不安定さの大きい広域分散環境にあると考えられる．今回の評価は，手法の基本性能を知りたいという動機から，単純な負荷，故障モデルについて PC クラスタ上で評価を行った．今後は，より実践に近い広域分散環境をモデリングした評価，あるいは広域分散環境上そのものでの評価を行っていきたい．特に，今回は PE の故障に関して，ジョブを受け取った瞬間に故障が発生するという単純なモデルを採用している．今後は，より一般的な，PE の故障が確率的に発生するモデルを仮定した評価を行っていきたい．

現在，広域分散環境上で RPC（Remoto Procedure Call）ベースのプログラミング環境やマスタ/ワーカ処理を実現するミドルウェアやシステムが開発されている[7)~10)]．本手法とこのようなミドルウェア，システムとの融合についても考えていきたい．また，Condor[3)] に代表される耐故障をミドルウェアレベルで実現するシステムとの比較評価，あるいは組合せについても考えて行きたい．

7. 関 連 研 究

アプリケーションレベルで耐障害性能を実現する関連研究を 3 つあげる．また，動的に負荷が変わる計算機環境に対するジョブスケジューリングアルゴリズムの提案と理論的考察を行った研究を紹介する．

Iamnitchi らによる並列分枝限定法アルゴリズムは，マスタを持たないタイプの並列プログラムである[4)]．広域分散環境で大規模な計算ノードを使って計算する状況を想定している．ジョブは各計算ノードに割り付けられ，近傍のノードどうしは情報交換をし，ジョブを融通しあって負荷の均等化を図っている．1 つのジョブが複数の計算ノードで重複して実行されることでノード故障に備えている．

Li らは Computational Replication という方法でマスタ/ワーカ処理に耐障害性能を加えている[5)]．これは，1 つのジョブを複数の計算ノードで実行し最も早く返された結果を採用するというもので，我々の方法とベースとなる考え方は共通である．彼らのメインのターゲットはモンテカルロ法である．モンテカルロ法が計算に乱数を用いるという特徴を用いて N-out-of-M スケジューリング（ジョブを M 個実行し，先着 N 個を回収したら先に進む）という方法で耐障害性能を実現している．

SETI@home[11)] もジョブを多重化して実行し，その中から正しい答えを得るという意味では本手法と関連する研究といえる．彼らのプロジェクトは悪意のある計算ノードを想定しなければならないため M 個投げたうちの N 個同じ答えが返ってきた時点でジョブが完了したと見なしている．

これらの研究と本手法の相違点は，ジョブに優先度を付加し，優先度に応じてジョブの多重度を決定しているところである．

Fujimoto らは，計算機の負荷が動的に変化する環境上で独立したジョブをスケジューリングする際に，ジョブの動的な多重化および不要になったジョブを停止するアルゴリズムを提案している[12)]．彼らは，TPCC

という指標でスケジューリングアルゴリズムの性能を評価しており，ジョブ数に対してマシン台数がある程度大きければ，提案アルゴリズムは最適なスケジューリングと性能がほぼ変わらないことを理論的に示している．取り扱っている問題が，ジョブが独立，処理時間が一定という仮定はあるが，本稿で述べた手法に対して多重化リストの動的な変化を導入するうえで，また，手法の理論的な解析を行ううえで有用な結果が示されている．

8. ま と め

本稿ではマスタ/ワーカモデルに基づく並列分枝限定法に耐故障/耐高負荷性能を持たせる手法について述べ，その基本性能の評価を行った．手法のポイントは 2 点ある．1 点目は，ジョブを多重化して実行し，ジョブに優先度を付けて多重度を制御していることである．これによりワーカの故障に備えられるほか，負荷の高いワーカが存在した際にも計算性能の著しい低下を防ぐことができる．2 点目は，ジョブの終了をもってジョブプールからジョブを削除する枠組みを採用していることである．ジョブの優先度の相対的順位が動的に変化するので，優先度が低く 1 ワーカで実行されて停滞/停止しているジョブも，いずれは多重化されて実行されるようになる．

実験では，本手法の基本的な挙動について示した後，耐高負荷機能や耐故障機能が機能していることを確認した．本稿の結果は，大規模な PE 数を利用して並列分枝限定法を解く際に，高速化のために利用していた PE の一部を，少ないオーバヘッドで冗長性のために転用できることを示している．今後は，今後の課題で述べた事項に取り組み，本手法のさらなる評価および実用化を図っていきたい．

謝辞 本研究を進めるにあたり貴重なご意見をいただいた，産業技術総合研究所の田中良夫研究員，中田秀基研究員，首藤一幸研究員に深く感謝いたします．また，本稿の構成に対して有益な助言および参考となる研究を提示していただいた査読者の方々に感謝いたします．

参 考 文 献

1) Foster, I. and Kesselman, C.: *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann (1988).
2) 南谷 崇：フォールトトレラントコンピュータ，オーム社 (1991).
3) Condor Project Homepage. http://www.cs.wisc.edu/condor/
4) Iamnitchi, A. and Foster, I.: A Problem-Specific Fault-Tolerance Mechanism for Asynchronous, Distributed Systems, *Proc. International Conference on Parallel Processing 2000*, pp.4–14 (2000).
5) Li, Y. and Mascagni, M.: Improving Performance via Computational Replication on a Large-Scale Computational Grid, *Proc. CCGrid 2003*, pp.442–448 (May 2003).
6) 武宮 博，田中良夫，中田秀基，関口智嗣：Ninf-G2：大規模 Grid 環境での利用に即した高機能，高性能 Grid RPC システムの実装と評価，SACSIS2004, pp.69–76 (2004).
7) 中田秀基，田中良夫，松岡 聡，関口智嗣：GridRPC を用いたタスクファーミング API の試作，情報処理学会研究報告，2003-HPC-96, pp.61–66 (2003).
8) 首藤一幸，大西丈治，田中良夫，関口智嗣：計算機資源の流通および集約のための P2P ミドルウェア，情報処理学会論文誌：コンピューティングシステム，Vol.45, No.SIG6(ACS 6), pp.208–222 (2004).
9) 秋山智宏，中田秀基，松岡 聡，関口智嗣：グリッド環境に適した並列組み合わせ最適化システム jPoP における分枝限定法の実装，*SPA 2003* (2003).
10) Goux, J.-P., Kulkarni, S., Linderoth, J. and Yoder, M.: An Enabling Framework for Master-Worker Applications on the Computational Grid, *Proc. 9th IEEE Symposium on HPDC*, pp.43–50 (2000).
11) SETI@home: Search for Extraterrestrial Intelligence at Home. http://setiathome.ssl.berkeley.edu/
12) Fujimoto, N. and Hagihara, K.: Near-Optimal Dynamic Task Scheduling of Independent Coarse-Grained Tasks onto a Computational Grid, *Proc. International Conference on Parallel Processing 2003*, pp.391–398 (2003).

(平成 16 年 1 月 31 日受付)
(平成 16 年 5 月 9 日採録)

久保田和人（正会員）

昭和 39 年生．昭和 63 年早稲田大学理工学部電子通信学科卒業．平成 5 年同大学大学院理工学研究科博士課程修了．同年（株）東芝入社．平成 7 年 10 月より平成 10 年 9 月まで技術研究組合新情報処理開発機構に出向．工学博士．ハイパフォーマンスコンピューティング，データマイニングに関する研究に従事．電子情報通信学会会員．

仲瀬 明彦（正会員）

昭和 36 年生．昭和 61 年早稲田大学大学院理工学研究科修士課程修了．同年（株）東芝入社．平成 5 年 4 月より平成 7 年 3 月まで（財）新世代コンピュータ技術開発機構に出向．並列処理，データマイニングに関する研究に従事．電子情報通信学会会員．

正　誤

情報処理学会論文誌：コンピューティングシステム Vol.45，No.SIG6 (ACS 6) pp.171–175 に掲載されました

論文題目：LISTVEC 指示行を使った多粒子シミュレーションの大規模化
—主メモリを節約し，かつ高速化を可能にする一つの方法
著 者 名：杉山　徹，寺田直樹，村田健史，大村善治，臼井英之，松本　紘

につきまして，171 ページ最終行において記述に誤りがありましたので，下記のように訂正致します．

誤）NEC 社製の SX–5 である．
↓
正）NEC 社製のベクトル計算機 HPS（High-speed Scientific Processor）である．