社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2009－MPS－73 (58)
2009／3／6

ピュア P2P ネットワーク上における分枝限定法の並列化 －Churn 発生時の耐故障性の研究－

柳 崇之, 品野 勇治

東京農工大学

分枝限定法は組合せ最適化問題に対する厳密解法として知られ，近年ではその並列化が多くの研究者によって研究されている．しかし，現在研究されている並列分枝限定法の多くはマスター・ワーカー方式によるものであり，マスターとなる計算ノードが存在しないピュア P2P ネットワーク上での実現には適していない．また P2P ネットワークでは，ネットワークを構成する計算ノードの離脱・参加が継続的に発生するため，分散処理を実現する際には耐故障性が重要な問題となる．本稿では，ピュア P2P ネットワーク上での分枝限定法の並列化，耐故障化手法について述べる．エミュレーターによる実験の結果，10 台の計算ノードによるピュア P2P ネットワーク上において，分枝限定法の並列化を行うとともに，その耐故障性を確認することができた．

Parallelization of Branch and Bound Algorithms on Pure P2P Network －Fault Tolerant Mechanism to Handle Churn－

Takayuki Yanagi, Yuji Shinano

Tokyo University of Agriculture and Technology

A Branch and Bound algorithm is a well known method to solve combinatorial optimization problems.Many researchers study the parallelization by using Master-Worker paradigm.However,such ways are not appropriate for pure P2P Networks where none of master node exists.In addition,fault tolerance is an important issue,because it happens continuously that process nodes join to or disjoin from P2P networks.In this paper,we propose a fault tolerant parallelization of branch and bound algorithm.The result of experiments by using emulator shows that our mechanisms are effective in parallelization and fault tolerance on the pure P2P network which includes 10 nodes.

1 はじめに

Peer-to-Peer(P2P) ネットワークは，ネットワーク上に接続された計算機 (ノード) 同士が対等に通信を行う自律分散ネットワークであり，高価な中央サーバーなどを必要とせずに，大規模なネットワークを構築することができる．近年，P2P ネットワーク上の各ノードに経路情報として分散ハッシュテーブル (DHT) を持たせることにより，ネットワーク上の位置に依存せず，効率的にデータを検索する手法が注目を集めている．分散ハッシュテーブルは，従来の P2P ネットワークでは探索が難しいと考えられている大規模なネットワークにおいても，ネットワーク内の情報を効率的に検索することができる仕組みであり，分散処理技術への応用も期待されている．

しかし P2P ネットワークにおいては，ネットワークへの参加・離脱が容易に行われるため，クラスタマシンなどを用いた分散処理以上に，計算ノードのネットワークからの離脱が発生しやすい．この現象は Churn とよばれ，P2P ネットワーク上での並列分枝限定法では，計算の途中で Churn が発生しても最適解を求めることができる仕組みが必要となる．並列分枝限定法における耐故障化手法の研究は多くなされているが，その多くはマスター・ワーカー方式によるもの [1] で，P2P ネットワーク上での耐故障化についての研究はまだ少ない．

本稿では，DHT を全ノードの共有スペースとして利用することで，ピュア P2P ネットワーク上での分枝限定法の並列手法を提案するとともに，その耐故障化手法についても述べる．具体的には DHT 自体に耐 Churn 手法 [2] を実装し，各ノードが解いている問題の情報を DHT 上へ登録しておく．ノードがネットワークから離脱した場合には，登録してある子問題を他のノードが解き直すようにすることで，耐故障性を得る．評価実験では Overlay Weaver[3] というオーバレイ構築ツールキットのエミュレータ機能を用いて，10 台の計算ノードによるピュア P2P ネットワーク上での並列分枝限定法のエミュレーションを行い，各ノードへの負荷分散の性能を調べた．また計算中に故意に Churn を発生させることで，耐故障性の検証も行った．

2 関連研究

ハッシングを利用した耐故障並列化手法としては野澤らの研究 [4] や，横山らの研究 [5] が関連研究として挙げられる．これらは，予めハッシュ空間において各ノードが計算を担当する空間を決めておき，子問題毎にハッシュ値を求めて担当ノードに割り振って行くことで並列処理を実現している．また，子問題のハッシュ値をキーとしてその計算結果をテーブルに保存しておくことで，複数の計算ノードが探索の過程で共通の部分問題を重複して解くことのないようにしている．多くの子問題の計算結果が各ノードのハッシュテーブルに保存されてるこ

とから，故障が検知された際には元の問題を初めから解き直しても大きな計算コストにはならないだろうという考え方である．しかしこれは計算を終えたすべての子問題の結果を記録しておく必要があり，また Churn のように頻繁に発生するノードの離脱に対しては十分対応できるとは言えない．

中川ら [6] は複数のクラスタからなるグリッド上での並列組合せ最適化アプリケーションの開発を容易に行うためのシステム jPoP を提案し，分枝限定法のためのクラス群の設計・実装を行っている．

3 並列化手法

並列処理を行う各計算ノードは，以下のような手順で分枝限定法の並列化を実現している．ここで扱う子問題とは分枝限定法の分枝操作によって，元の問題の実行可能領域を分割して得られた部分問題を指す．

1. 初めに指定ノードが分枝操作のみを数回行って，子問題を生成．
2. 子問題の取得
3. 分枝限定法の適用
4. 暫定解の更新
5. DHT への子問題の登録

3.1 子問題の取得

各ノードは，ハッシュテーブル上に子問題を登録するための共通のキーを $key1, key2, \ldots, keyN$ のように複数持っている．各ノードは自分のプールから終端していない子問題を取り出すか，DHT 上からまだ解かれていない子問題を取得して子問題を解きはじめる．自分のプールが空で，DHT 上から子問題を取得する際には，複数のキーの中からランダムに 1 つのキーを選択し，登録されている子問題を取得する．選んだキー $keyX$ に子問題が登録されていなかった場合には，$key\{(X+1) \bmod N\}$ で同様の操作を行う．子問題を取得するか，全てキーでこの操作を行うまでこの操作を繰り返す．

3.2 分枝限定法の適用

子問題を取得したノードは，取得した子問題に対して分枝限定法を適用する．各ノードには子問題を取得してから分枝操作を行うことができる回数に制限 (Branch Limit) を設け，生成された子問題がすべて限定操作によって終端されるか，制限回数だけ分枝操作を行った場合にはその時点で分枝限定法の適用を止める．

3.3 暫定解の更新

DHT には分枝限定法によって得られた暫定解が登録されており，各ノードは分枝限定法を Branch Limit まで適用する前後で暫定解の更新作業を行う．更新作業では，DHT 上に登録されている暫定解と自身が持っている暫定解を比較し，自身がより良い暫定解を持っている場合には DHT 上の暫定解を書きかえる．また，更新した暫定解と自分のプール内の子問題の上界値を利用して限定操作を適用する．

3.4 子問題の登録

Blanch Limit まで分枝限定法を適用し終えた時にまだ終端していない子問題が残っている場合には，その子問題を上界値の高い順にソートし，自分のプールと DHT へ交互に登録していく．各ノードが子問題を確保する数には上限 (Pool Limit) を設け，プールの子問題の数が上限一杯の場合には残りの問題を確保せずにすべて DHT へ登録する．これによって，最適解に辿り着く子問題などの，限定操作が適用されにくい子問題を取得したノードへの負荷が集中することを避ける．

4 耐故障化手法

文献 [2] より，0.15 秒毎という非常に短い間隔でノードの離脱・参加が発生する環境においても，95〜99% と高い精度で DHT 上の値を保証することができていた．そこで本稿では文献 [2] に示されている耐 Churn 手法を用いることで，DHT に登録してある子問題を保証できると仮定する．次にこれを利用して，故障などによって各ピアが最適解に行きつく子問題を持ったままネットワークから離脱してしまい，分枝限定法で最適解を求めることができなくなってしまうことを防ぐ．

4.1 基本方針

並列分枝限定法における耐故障は，ピュア P2P ネットワーク上であっても基本的にはマスター・ワーカー方式によるものと変わらない．即ち，ピアの離脱によって解を出す前に消失してしてしまった子問題を再度別のピアが解き直すことで耐故障性を得る．このために必要な機能として次のようなものが考えられる．

1. 各ピアが持つ子問題の情報保持
2. 故障の検知
3. 子問題の再割り当て

マスター・ワーカー方式では，これらの機能はすべてマスターの計算ノードが持っている．しかしマスターのような特別なノードを持たないピュア P2P ネットワークでは，これらの機能をなんらかの方法で実現しなければならない．そこで提案手法では，子問題の共有プールとして利用している DHT を用いて上記の機能を実現する．具体的には DHT 上に，現在解いている子問題を置いておく running プールを用意することで各ピアが持っている子問題の情報を管理する．また，各ノードは running プールを監視するチェッカーをスレッドで実行する．

4.2 子問題の取得

各ノードは子問題の取得に次のような操作を行う．以下で扱う Get とは，3.1 に示した子問題を取得するための一連の操作のことを指し，Put とは DHT 上へ子問題を登録する操作のことを指す．

Step.1 $keyX$ から子問題 P_k を Get
Step.2 $keyX$ に P_k が残っている場合は,running プールへ P_k を Put
Step.3 $keyX$ に P_k が残っている場合には $keyX$ から P_k を Remove し,P_k を解き始める

Step.1 において，複数のノードが同時に子問題の Get を行う可能性があるが，Step.3 において P_k の Remove を行ったノードのみが P_k を解きはじめることで，複数のノードが同じ子問題を解きはじめることを防いでいる．また，Step.2 において複数のノードから同じ子問題が running プールへ Put される場合があるが，DHT は同じ key に同じ value を複数回 Put されても，value が重複してハッシュテーブルに登録されることはないため，同じ子問題が複数個 running プールへ登録されることはない．この処理では P_k を running プールへ Put した後に Remove 操作を行うため，P_k を取得したノードがネットワークから離脱しても，P_k が完全に DHT 上から消失することはない．P_k を解いていたノードがネットワークから離脱した場合には，4.4 に示すチェッカーのはたらきによってランダムに選択された $KeyX$ へ Put される．

4.3 子問題の登録

3.3 では生成された子問題のすべてを DHT へ投げるのではなく，一部の終端していない子問題を自分のプー

ルに保持していた．しかしこれではすべての子問題を管理することができないため，通常の Put 以外に，プールへ保持する子問題を running プールへ Put するようにする．

また，各ノードは，子問題 P_k に一定回数分枝限定法を適用する前に P_k を一時的に保存しておく．各ソルバは P_k に分枝限定法を適用することで得られた子問題に対して，Put の処理を行ったの直後に P_k を running プールから削除する．P_k から分枝して得られた子問題を DHT 上へ Put してあるため，P_k はこれ以上保存しておく必要がないためである．これによって必要のない無駄なコピーがいつまでも DHT 上へ残らないようにしている．

4.4 チェッカーの行う手続き

各ノードはそれぞれ，チェッカーとよばれる running プール監視用のスレッドを作成する．チェッカーは running プールにある子問題の TTL [*1] を定期的にチェックする．TTL は running プールへ Put する際に指定され，TTL が 0 になってしまうと子問題がハッシュテーブルから削除されてしまう．本稿では，TTL が一定値以下になった子問題，即ち running プールへ Put されてから一定時間以上消去されずに残った子問題を見つけた場合，その子問題を解いているノードが故障したとみなす．子問題 P_k の TTL が一定値以下となっているのを見つけたチェッカーは，ランダムに 1 つのキーを選択して P_k を Put したのち，running プールから P_k を削除する．

5 評価実験

評価実験では Overlay Weaver が提供する DHT の API を利用することで，分散ハッシュテーブルによって情報の共有を行い，ナップサック問題を並列分枝限定法によって解くアプリケーションの開発を行った．このアプリケーションには 4 章で提案した耐故障アルゴリズムも実装されている．実験では，Overlay Weaver のエミュレータを用いて 10 台のピアによるピュア P2P ネットワーク上での並列分枝限定法のエミュレーションを行った．ここでは提案する耐故障手法によって，最適解が正しく保証されるかを調べると共に，Overlay Weaver のメッセージングサービスを利用してネットワーク上の通信回数と各ピアの処理負荷を調べた．

5.1 パラメータ

評価実験を行うために用いた実験環境，および各パラメータについて述べる．

■実験環境 実験環境を表 1 に示す．

表 1 実行環境

OS	Linux-2.6
CPU	Pentium Ⅲ 1.0GHz
メモリ	2GB
JDK	JDK 1.6.0
Overlay Weaver	Version 0.8.8

■並列分枝限定法のパラメータ 並列分枝限定法の各パラメータの値を 2 種類設定してまとめたものを表 2 に示す．Type A は並列化の粒度が非常に細かく，並列化を行っているノード間でのより良い負荷分散が期待される．一方 Type B は Type A に比べて並列化の粒度が粗く，問題の解くための通信回数や耐故障処理のためのメモリやストレージを少なくできることが期待される．

表 2 並列分枝限定法のパラメータ

	Type A	Type B
初期分枝回数	6	4
Branch Limit	1000	5000
Pool Limit	10	100

■Churn Churn は，構造化オーバーレイネットワークを構成す 10 台のうち，指定した 1 台のピアを除く 9 台のピアから離脱ピアを乱数を用いて選択し，ピアを離脱させた直後に新たなピアをネットワークに参加させる．この処理を 1 秒毎に繰り返し行い，すべてのピアのアプリケーションが終了するまで Churn を起こし続けるエミュレータのシナリオを作成した．DHT に加入しているピアの数を開始時の 10 台に保つこの Churn モデルは，首藤の実験 [2] と同じものである．

5.2 実験結果

アイテム数 1000，3000，5000 個のナップサック問題に対してそれぞれ 3 種類の難易度の問題を用意し，計 9 種類の問題に対して表 2 に示した 2 つのタイプで実験を行った．ナップサック問題の生成には D.Pisinger 氏の作成したプログラム [7] を利用した．各タイプにおいて，提案した耐故障手法によって,Churn 発生下で最適解を正しく求めることができるかを調べたところ，耐故障処理を未実装の場合に最適解が得られなかった 5 種類の問題において，提案手法によって最適解を得ることができた．しかし今回の実験では，アイテム数が多いか難易度が高い 7 種の問題で，エミュレータがメモリオーバーをおこしてしまい問題を解くことができなかった．

5.2.1 通信回数

Overlay Weaver のメッセージングサービスを用いて各設定における問題毎のメッセージの通信回数を調べた．結果を図 1, 図 2 に示す．ここであげる通信回数は，Put, Get の際のルーティングのために送信されるメッセージや，ノードのネットワークへの参加要求など様々なメッセージの総数である．図中横軸の数字はナップサック問題の種類を表しており，“ナップサック問題のアイテム数 - 問題のタイプ”を表している．

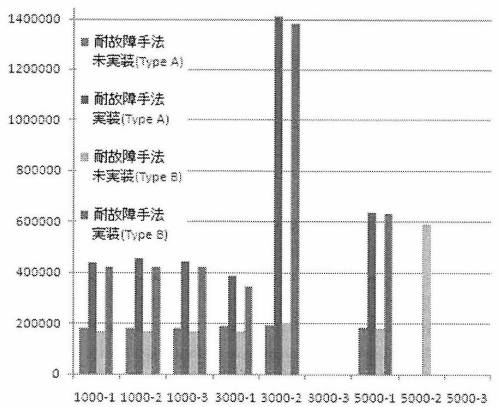

図 1 Churn 未発生時の通信回数

[*1] Time to Live:DHT に登録した内容が生存していられる残り時間.

表 3　アイテム数 3000 の問題における各ノードの負荷

ノード (ソルバ) の番号	耐故障手法未実装 (Type A)	耐故障手法実装 (Type A)	耐故障手法未実装 (Type B)	耐故障手法実装 (Type B)
Sol.1	29(3434)	30(3434)	44(5510)	44(5510)
Sol.2	47(2924)	30(1000)	64(5000)	45(6142)
Sol.3	29(1000)	45(1008)	48(5000)	60(10000)
Sol.4	42(1000)	48(2000)	58(10844)	57(7016)
Sol.5	45(1000)	43(996)	49(5000)	49(8012)
Sol.6	22(1000)	31(1000)	45(7912)	44(4826)
Sol.7	44(2438)	23(984)	55(10000)	53(5000)
Sol.8	40(1000)	47(2436)	63(7136)	57(10052)
Sol.9	46(2000)	46(3012)	47(5000)	43(5000)
Sol.10	27(1000)	41(1000)	43(7920)	54(5213)

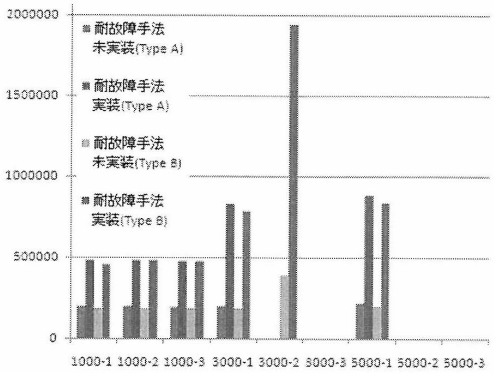

図 2　Churn 発生時の通信回数

5.2.2　負荷分散

各ノードの分枝限定法の実行回数および，分枝操作の適用回数を問題毎にまとめたものの一部を表 3 に示す．ここでの「負荷」とは，ノード毎の分枝限定法の実行による負荷を表しており．表中の数字は“分枝限定法の実行回数 (分枝回数)”を示している．

6　考察と今後の課題

並列化による負荷分散に関しては，アイテム数の少ない問題，難易度の低い問題では負荷の偏りが見られたが，表 3 のようにある程度の問題規模になると，その偏りは小さくなっている．また評価実験により，提案する耐故障手法によって Churn 発生時においても分枝限定法による最適解を保証することができた．しかし耐故障化手法の実装により計算ノード間の通信回数は最大で約 7 倍にまで増加し，それ以外の問題でも平均して 2 倍〜3 倍に増加してしまった．DHT の耐 Churn 手法によって，DHT 上には実際に Put された個数の少なくとも 3 倍以上の子問題が登録されてしまうため，問題が大きくなって Get,Put の頻度が増すにつれて，DHT 上の子問題を保存するためのメモリの必要量も急激に増加してしまう．そのためアイテム数を大きくした場合に，1 台の計算機上でエミュレーションを行うことができなくなってしまったのだと考えられ，この必要メモリ容量や通信量の削減が今後と課題として残った．また実環境への応用を考える場合には，ネットワークの帯域幅や信頼性，Churn の発生頻度などを考慮して各パラメータの値を定めるための実験が必要となる

参考文献

[1] 川上 健太, 合田 憲人, ”並列分枝限定法における耐故障アルゴリズムの評価”, 情報処理学会研究会報告: ハイパフォーマンスコンピューティング, Vol.2005, No.81(HPC 103), pp.49-54. 2005 年.

[2] 首藤 一幸, ”下位アルゴリズム中立な DHT 実装への耐 churn 手法の実装”, 情報処理学会論文誌:コンピューティングシステム, Vol.49, No.SIG2(ACS 21), pp.1-9, 2008 年.

[3] 首藤 一幸, 田中 良夫, 関口 智嗣, ”オーバーレイ構築ツールキット OverlayWeaver”, 情報処理学会論文誌:コンピューティングシステム, Vol.47, No.SIG12(ACS 15), pp.358-367, 2006 年.

[4] 野澤 康文, ”分散ハッシュに基づく大規模探索問題の耐故障並列化手法”, 情報処理学会論文誌:プログラミング, Vol.47, No.SIG16(PRO 31), pp.87-94, 2006 年.

[5] 横山 大作, 田浦 健次郎, 近山 隆, ”ハッシングに基づく大規模探索問題の耐故障分散処理手法”, 情報処理学会論文誌:プログラミング, Vol.48, No.SIG 4(PRO 32), pp.1-13, 2007 年.

[6] 中川 伸吾, 中田 秀基, 松岡 聡, ”並列組合せ最適化システム jPoP の分枝限定法の実装”, コンピュータシステム・シンポジウム論文集, pp.85-92, 2004 年.

[7] David Pisinger, ”David Pisinger's optimization codes”, http://www.diku.dk/hjemmesider/ansatte/pisinger/codes.html, 2008 年 9 月 1 日参照.