日本ソフトウェア科学会第 29 回大会 (2012 年度) 講演論文集

m-bridgeによる分割を適用した木構造処理のHadoop上での評価

川村 高之　松崎 公紀

木構造は探索アルゴリズムや XML 文書等で用いられる有用なデータ構造であり，近年では木構造を利用しているデータの大規模化が進んでいる．大規模データ処理において，データを分割して並列化を行い処理の高速化することが重要である．本研究では，一般的な木構造に対して m-bridge を適用した木構造処理について調査する．m-bridge は単純なルールで任意の形の木を分割できる手法である．m-bridge を用いて分割した木構造データを大規模クラスタ向け並列計算フレームワークである Hadoop を用いて処理する際の実行時間について，16 台構成のクラスタ上で複数の木データに対して実験を行い評価する．

1　はじめに

木構造は XML などの構造を持った文書や探索アルゴリズム [2] などで用いられる有用なデータ構造である．また，XML データベースのように木構造を持つ大規模データが扱われるようになり，またその規模は拡大している．

データが 1 台のハードディスクに収まらず，また，現実的な時間で処理ができないような大規模データを扱うためには，データを分散し並列処理を行う必要がある．大規模クラスタ上でそのような並列データ処理を実現するためのフレームワークのひとつに Hadoop [15] がある．Hadoop においては，データは分散ファイルシステム上に分散して置かれる．また，ユーザは，MapReduce プログラミングモデルによって比較的容易に並列プログラムを作成することができる．

これまで Hadoop ／ MapReduce に関する多くの研究や応用では，その要素間に関係があまりないデータを対象としていた．要素間に関係がなければ，データを自由に分割・分散することができるため，その並列化は容易である．しかし木構造の場合には，そのノード間に親子関係や兄弟関係が存在する．したがって，木構造データの場合には，好き勝手にデータを分割・分散することはできない．また，木構造は不均等になりうるため，ルートノードから分割していくような単純な分割手法では負荷分散の点で問題となる．

そこで本研究では，m-bridge による木の分割手法を適用した並列処理について，その性質を調査する．m-bridge による木の分割手法は木のノード数に関する簡単な計算によって分割するもので，分割された各部分が高々m ノードからなるなどいくつかの良い性質を持つ．子の数が任意である木構造 (薔薇木) に対して m-bridge を適用した場合と，同じ木を二分木に変換して m-bridge を適用した場合について，その分割の性質を調査する．また，m-bridge を用いて分割した木構造データを Hadoop を用いて処理した際の実行時間について，16 台構成のクラスタを用いて評価する．

本論文の構成は以下の通りである．第 2 節では m-bridge による木の分割手法について述べる．第 3 節では分散処理フレームワークである Hadoop を構成する HDFS と MapReduce について述べる．第 4

Evaluation of Tree Processing based on the m-bridge Technique over Hadoop.

Takayuki Kawamura, 高知工科大学大学院工学研究科, Graduate School of Engineering, Kochi University of Technology.

Kiminori Matsuzaki, 高知工科大学情報学群, School of Information, Kochi University of Technology.

節では m-bridge の分割の性質の調査について述べる．第 5 節では Hadoop を用いた実験について述べる．第 6 節では関連研究についてを述べ，第 7 節で本論文をまとめる．

2 m-bridge

木を分割する最も簡単な手法は，目的の分割数もしくはノード数となるまで再帰的にそのルートノードで分割するというものである．しかしこの手法では，木が不均等である場合にうまく分割することができない．本節では，m-bridge による木の分割手法 [5] について述べる．m-bridge を用いた木の分割は，部分木のノード数による簡単な計算によって求めることができ，また分割された各部分のノード数が m 以下であるなどの良い性質を持つ．

以下では，木のノード x に対して，x.size によってそのノード以下の部分木のノード数を表すものとする．m-bridge を用いた木の分割では，この部分木のノード数に着目して木を分割する．m は分割のパラメータとして与えられた数である．

定義 1 木のノード x が次の 2 つの条件を満たすとき，ノード x は m-criticial であるという．

- ノード x は葉ではない．
- ノード x のすべての子 c について，
$$\left\lceil \frac{x.\text{size}}{m} \right\rceil > \left\lceil \frac{c.\text{size}}{m} \right\rceil$$
が成り立つ． □

m-bridge による分割は，すべての m-critical なノードの直下で木を分割するというものである．図 1 に m-bridge による木の分割の例を示す．図において，ノード a は m-critical なノードである．なぜならば，$\lceil$a.size/3$\rceil = \lceil 9/3 \rceil = 3$ であり，その 2 つの子について，$\lceil$b.size/3$\rceil = \lceil 5/3 \rceil = 2$ と $\lceil$c.size/3$\rceil = \lceil 3/3 \rceil = 1$ より定義 1 の数式を満たすからである．

m-bridge を用いた木の分割は並列計算の観点で良い性質を持つ．以下にそのいくつかを示す [5] [11] [7]．

補題 1 m-bridge によって分割された各部分は，そのノード数が高々m である． □

補題 2 m-bridge によって分割された各部分には，m-critical なノードは高々1 つしかない． □

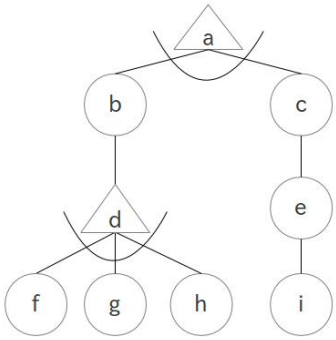

図 1 m-bridge により木を分割する例 ($m = 3$)．図中の △は m-critical なノード，○はそれ以外のノードを表す．

補題 3 N ノードの木を m-bridge によって分割するとき，m-critical なノードの数は $2N/m - 1$ 以下である． □

補題 4 N ノードの**二分木**を m-bridge によって分割するとき，m-critical なノードの数は $(N/m - 1)/2$ 以上である． □

補題 3 により m-critical なノードの数は m を用いた式で抑えられるが，一般にはそのノードがいくつ子を持つか分からないため分割された部分の数を式で抑えることはできない．一方，二分木の場合には，分割された部分の数は $4N/m - 1$ 以下と抑えることができる．

本節の最後に，木および m-bridge によって分割された木の表現について述べる．本研究では子の数が任意である木（薔薇木）を扱うため，木の表現法として XML に類似した形式を用いた．具体的には，XML の開きタグにあたるものとしてそのノードの値を，閉じタグにあたるものとして-1 を用いる．図 2 に図 1 の木構造を表現したものを示す．m-bridge によって分割した木のデータを表現するにあたり，分割された各部分の関係も保持する必要がある．したがって，分割された部分の関係を示す**構成木**と，各部分である**分割木**を分けて保存する．構成木は，各部分に振られた連番を用いて XML に類似した形式で表現する．一方，分割木のデータは，それに振られた連番の後にその部分に含まれるノードを XML に類似した形式で表現する．各ノードは，そのノードの値の他に，そ

```
a b d f -1 g -1 h -1 -1 -1 c e i -1 -1 -1 -1
```

図 2　図 1 のテキスト表現

構成木

```
0 1 2 -1 3 -1 4 -1 -1 5 -1
```

分割木

```
0:a,T -1,F
1:b,F d,T -1,F -1,F
2:f,F -1,F
3:g,F -1,F
4:h,F -1,F
5:c,F e,F i,F -1,F -1,F -1,F
```

図 3　図 1 の分割のテキスト表現

のノードが m-critical なノードであるかを表すマークを持つ．図 3 に図 1 の分割されたデータを表現したものを示す．

3　Hadoop

Hadoop [15] は，大規模なデータに対して効率的に並列処理を行うための分散処理フレームワークのひとつである．特に，その MapReduce フレームワークは，Google の MapReduce フレームワーク [3] のオープンソース実装である．本節では，Hadoop を構成する分散ファイルシステム HDFS と MapReduce について述べる．

分散ファイルシステム HDFS は，コモディティマシンで構成される大規模クラスタ上の分散ファイルシステムである．並列処理を行わなければならないようなデータは，データサイズが数テラバイト以上といったように非常に大規模なデータなることがあり，そのような場合には 1 つのストレージには格納できない．分散ファイルシステムでは，クラスタを構成する複数台の計算機上のストレージを利用して，全体で非常に大きなストレージを提供する．

MapReduce は，大規模クラスタ上の並列プログラムの実行環境であり，また並列プログラミングモデルでもある．MapReduce のプログラミングモデルを図 4 に示す．MapReduce では，map と reduce という 2

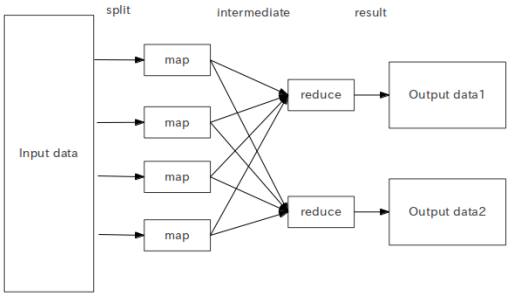

図 4　MapReduce モデル

つのフェーズに分割して処理が行われる．一般的な処理の流れとしては，map フェーズにおいて分割されたデータに独立に処理を行い，その後，reduce フェーズで map の結果を集約して出力する．map フェーズと reduce フェーズで行う処理に関してはプログラマ自ら記述する．各フェーズの入出力データの受け渡しはフレームワークによって自動で行われる．また，負荷分散や耐故障性の処理もフレームワークによって行われる．したがって，各フェーズ内の逐次的な処理を記述するだけで，クラスタ上で効率の良い並列処理を行うことができる．

4　m-bridge による分割の性質の調査

本研究ではまず m-bridge による分割の性質を調査した．乱数を用いて 10,000,000 ノードからなる薔薇木を 10 個生成し，さらにそれらの木を二分木に変換した．薔薇木の生成にあたって，各ノードの分岐数 k が平均 $\mu_1 = 12.0$ (木のノードの平均深さ 7.5) と分散 $\sigma_1 = 2.0$ の正規分布に従うようにし，さらにノード数 n の子の部分木が平均 n/k と分散 $n/2k$ の正規分布に従うようにした．(ただし，それぞれ 0 以下の値は 1 とした．) 二分木への変換については，各ノードの第 1 子要素を二分木の左の子，各ノードの次の兄弟を二分木の右の子となるように変換した [9]．ただし，文献 [9] で用いられている変換とは異なり，ダミーの葉要素は挿入していない．分割に用いる m の値は，100 と 1,000 と 10,000 の 3 つの値とした．

図 5 から 10 に，それぞれの分割によって得られた分割部分のノード数をヒストグラムにして示す．横の

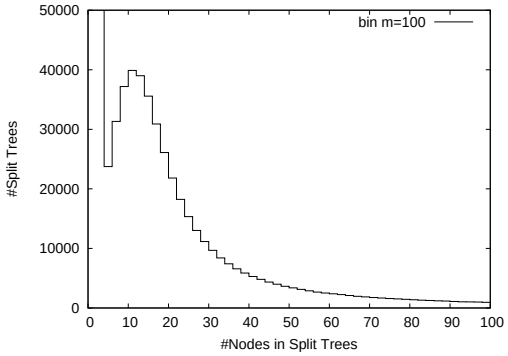

図 5　ノード数 $10,000,000$ の薔薇木に対して $m = 100$ で分割したときの分割部分の大きさ．ノード数が 2 以下の部分の数は 76126.3 である．

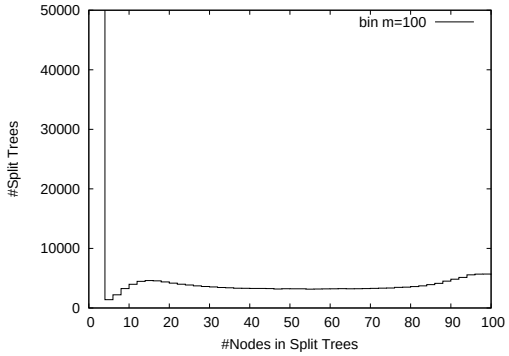

図 6　ノード数 $10,000,000$ の二分木に対して $m = 100$ で分割したときの分割部分の大きさ．ノード数が 2 以下の部分の数は 60143.9 である．

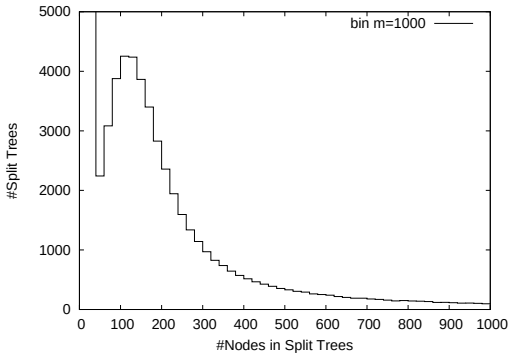

図 7　ノード数 $10,000,000$ の薔薇木に対して $m = 1,000$ で分割したときの分割部分の大きさ．ノード数が 20 以下の部分の数は 7277.1 である．

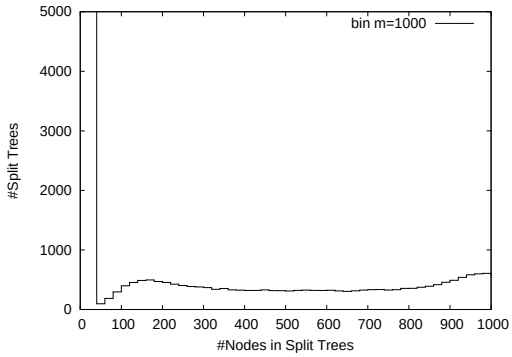

図 8　ノード数 $10,000,000$ の二分木に対して $m = 1,000$ で分割したときの分割部分の大きさ．ノード数が 20 以下の部分の数は 6124.2 である．

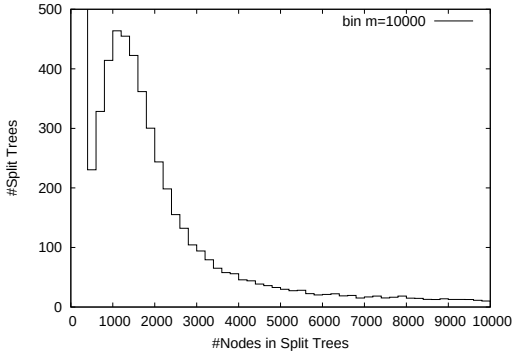

図 9　ノード数 $10,000,000$ の薔薇木に対して $m = 10,000$ で分割したときの分割部分の大きさ．ノード数が 200 以下の部分の数は 746.2 である．

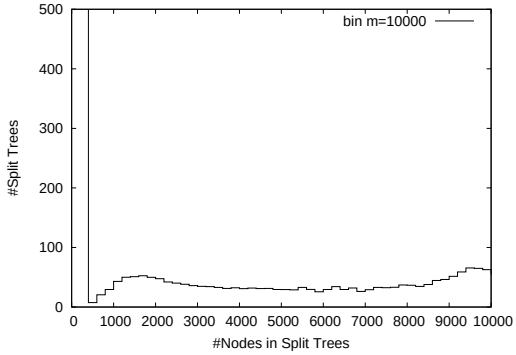

図 10　ノード数 $10,000,000$ の二分木に対して $m = 10,000$ で分割したときの分割部分の大きさ．ノード数が 200 以下の部分の数は 625.8 である．

スケールを調整するため，$m = 100$ のときにはノード数 2 ごと，$m = 1000$ のときにはノード数 20 ごと，$m = 10000$ のときにはノード数 200 ごとに集計している．

これらの結果より，いくつかの重要な知見が得られる．

1. m-bridge による分割で得られる部分のノード数の分布は，m の値を変えてもほぼ同じである．
2. 薔薇木に対する m-bridge の分割は，二分木に対する分割に比べて，分割数が大きい．例えば，$m = 10,000$ の場合，薔薇木からは平均 5,384，二分木からは平均 2,457 の部分が生成される．
3. 薔薇木に対する m-bridge の分割は，ノード数が少ない部分がより多く生成される．
4. ノード数が 1 である部分が多く生成されるが，その数は薔薇木と二分木でそれほど大きな差はない．

1 つ目の知見は著者らも予想していなかったものである．この知見により，m-bridge により分割した場合の性質を議論するにあたって，木の形状について考慮する必要はあるものの，分割に用いる m の値についてはあまり考察しなくて良いことが分かる．2 つ目と 3 つ目の知見は，薔薇木が分岐数が多いことから，m-critical なノードの下で分割する場合に分割数が多くなり，また定義 1 の条件の数式をより満たしやすいことから直観的に考察できる．4 つ目の知見から，ノード数が 1 となる部分に対して特別な実装を与えることで効率のより良い実装を得られることが示唆される．そのような実装のための方策については今後の課題である．

5 Hadoop 上での実験

m-bridge による木の分割を用いて木構造処理を行うプログラムの実行時間を Hadoop 上で計測した．

本実験では，木の高さを求める計算を用いた．この計算は計算コストが非常に小さく，Hadoop によるオーバヘッドが大きく出る．Hadoop に渡すデータは図 3 の形式を利用し，全体の構造を表す構成木については各ジョブが共通して持ち，各部分を表す部分は行ごとにジョブが実行されるようにした．用いた木構造データは，乱数によって生成されたノード数 10,000,000，高さが 7 である薔薇木とそれを二分木に変換したものとし，複数の m の値で分割したものを用いた．

表 1 実験機材 (16 台)

OS	Ubuntu12.04LTS 64bit
CPU	Intel(R) Core(TM)i5 CPU3.30GHz
memory	8GB
Java	Version6
Ethernet	1GB/s
Hadoop	Version 1.0.1

Hadoop の実行環境は表 1 に示される PC16 ノードから構成される PC クラスタである．Hadoop の設定は，

- `mapred.child.java.opts=-Xmx4096m`
- `dfs.block.size=1048576`

以外はデフォルトの値を用いている．

実験結果を図 11 と 12 に示す．図に示す処理時間は Hadoop にジョブを投入してから終了するまでの時間であり，Hadoop による各 PC へのデータの配布や map や reduce 以外の処理の時間も含んでいる．また，処理時間は 5 回計測を行いその平均を結果とした．

これらの結果より，薔薇木に対しても二分木に対しても台数効果による速度向上が得られていることが分かる．より詳しく見ると，二分木に変換した方が薔薇木に対する場合より高速であり，16 台の場合には 30〜40 秒程度の差がある．これは，m-bridge による分割において，薔薇木の場合により多くの部分に分割されることと，より小さな部分に多く分割されることがその理由であると考える．

次に，分割に用いる m の値は，実行結果にあまり影響を与えないことが分かる．分割数の数だけ map ジョブが生成されるが，Hadoop ではそのジョブの数は大きなオーバヘッドではないと考える．二分木の場合に $m = 500,000$ 以上の場合に実行時間が増えている．実行する PC の台数に依らず時間が増えており，この時間増加の理由は，部分の大きさが分散ファイル

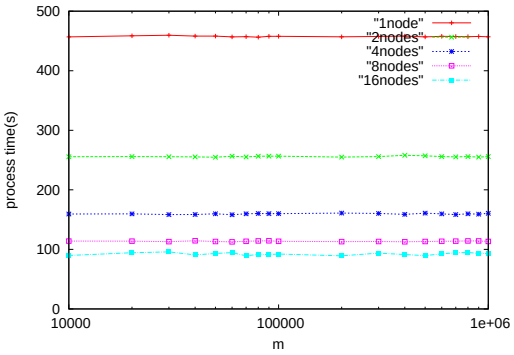

図 11　ノード数 10,000,000 の高さ 7 である根付き木のデータ

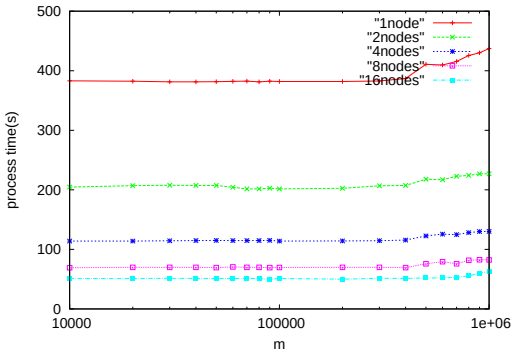

図 12　ノード数 10,000,000 の高さ 7 である根付き木を等価な二分木に変換した場合のデータ

システムのブロックサイズよりも大きくなったことによる読み込みのオーバヘッドであると考える．

6　関連研究

木構造に対する並列アルゴリズムのうち重要なものに Tree Contraction アルゴリズム [10] [1] がある．Tree Contraction アルゴリズムは，Rake と Compress と呼ばれる局所的な操作を並列に適用して計算を行うものであり，Abrahamson らの手法 [1] は二分木に対して簡単なスケジューリングにより並列計算が実現できる．PC クラスタのような分散メモリ環境では，m-bridge を用いて木を分割し，分割された部分に対する計算を局所化することでより効率良く計算することができる．Matsuzaki は，m-bridge による分割を用いて，二分木に対する効率的な並列計算パターン (並列スケルトン) の実装を与えている [8]．

子の数が任意であるような一般の木 (薔薇木) に対する並列処理では，XML に類似したシリアライズによってリスト形式となったデータに対する処理が提案されている [13] [6]．この手法では，データがリスト形式であることからその分割は容易に行うことができる一方で，計算の途中で通信によってやりとりされるデータが大きくなることがあるという欠点もある．XML に類似したシリアライズによる木構造処理の Hadoop 上の実装が，Emoto と Imachi によって与えられている [4]．また，薔薇木を二分木に変換した上で，二分木上の並列スケルトンを用いて計算を実現する手法も提案されている [9]．

大規模分散システムにおいて木構造処理を実現することを目的として Sarje と Aluru は新しいフレームワークを提案している [12]．そのフレームワークでは，select，treeCompute，combine と呼ばれる 3 つの基本処理をユーザが記述することで木構造全体の処理を並列化している．いくつかのアプリケーションにおいては良い性能が出ることが報告されているが，木データをどのように分散するかについて具体的な考察がなかった．

本研究の将来の目標は，大規模木構造データに対する計算パターンを Hadoop 上で実現することである．木構造に対する並列処理において有用なパターンとして並列スケルトン [14] [9] が提案されている．このような並列スケルトンを用いることで，XPath クエリの並列化 [17] や N 体問題シミュレーション [16] などのアプリケーションを効率良く実現することができる．

7　まとめ

本論文では，m-bridge を用いて木構造データを分割した際の性質，および，Hadoop を用いた木構造処理の実行時間について調査した．一般の木構造の場合，m-bridge を用いて木を分割すると小さな部分構造が多数生成されてしまうため，それが計算の際にオーバヘッドとなりうる．そのような木構造を二分木に変換した上で m-bridge により分割することで，より良い木の分割を得ることができる．実際に，16 台構成のクラスタ上で実験を行い，いずれの木構造の分割においても台数効果による速度向上が見られたが，

二分木に変換した上で m-bridge を適用する実装の方がより高速であった．

今後の課題としては，まず，極端に深い木や左右に偏りのある木といった，もっと多くの形の違う木に対して m-bridge を実行して，形の違いによる処理時間の変化に関してデータを取ることが挙げられる．また，本研究で実装した計算パターンをライブラリとして一般化し，XML 文書に対するクエリ処理などの具体的な応用例を実現したい．

Acknowledgments

This work is supported by JST (Grant Number 10102704; Japan) and ANR (project PaPDAS ANR-2010-INTB-0205-02; France), and JSPS KAKENHI Grant Number 10016934.

参 考 文 献

[1] K. R. Abrahamson, N. Dadoun, D. G. Kirkpatrick and T. M. Przytycka. A Simple Parallel Tree Contraction Algorithm. *Journal of Algorithms*, Vol. 10, No. 2, pp. 287–302, 1989.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. Introduction to Algorithms Third Edition. pp. 286–307, 2009.

[3] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *OSDI*, pp. 137–150, 2004.

[4] K. Emoto and H. Imachi. Parallel Tree Reduction on MapReduce. *Procedia Computer Science*, Vol. 9, pp. 1827–1836, 2012.

[5] H. Gazit, G. L. Miller and S.-H. Teng. Optimal tree contraction in EREW model. In *Proceedings of the Princeton Workshop on Algorithms, Architectures, and Technical Issues for Models of Concurrent Computations*, pp. 139–156, 1987.

[6] K. Kakehi, K. Matsuzaki and K. Emoto. Efficient Parallel Tree Reductions on Distributed Memory Environments. In *Proceedings of 7th International Conference (ICCS 2007)*, *Lecture Notes in Computer Science*, Vol. 4488, pp. 601–608, 2007.

[7] K. Matsuzaki. Parallel Programming with Tree Skeletons. 博士論文, 東京大学大学院情報理工学系研究科, 2007.

[8] K. Matsuzaki. Efficient Implementation of Tree Accumulations on Distributed-Memory Parallel Computers. In *Proceedings of 7th International Conference (ICCS 2007)*, *Lecture Notes in Computer Science*, Vol. 4488, pp. 609–616, 2007.

[9] K. Matsuzaki, Z. Hu and M. Takeichi. Parallel Skeletons for Manipulating General Trees. *Parallel Computing*, Vol. 32, No. 7–8, pp. 590–603, 2006.

[10] G. L. Miller and J. H. Reif. Parallel Tree Contraction and its Application. In *Proceedings of 26th Annual Symposium on Foundations of Computer Science*, pp. 478–489, 1985.

[11] J. H. Reif, editor. *Synthesis of Parallel Algorithms*. Morgan Kaufmann Publishers, 1993.

[12] W. Sarje and S. Aluru. A MapReduce Style Framework for Computations on Trees. In *Proceedings of 39th International Conference on Parallel Processing (ICPP 2010)*, pp. 343–352, 2010.

[13] F. E. Sevilgen, S. Aluru and N. Futamura. Parallel algorithms for tree accumulations. *Journal of Parallel and Distributed Computing*, Vol. 65, No. 1, pp. 85–93, 2005.

[14] D. B. Skillicorn. *Foundations of Parallel Programming. Cambridge International Series on Parallel Computation* series, Vol. 6, Cambridge University Press, 1994.

[15] T. White 著, 玉川竜司, 兼田聖士 訳. Hadoop. 株式会社オライリー・ジャパン, 2010.

[16] 佐藤重幸, 岩崎英哉. FMM への木スケルトンの適用. 第 14 回プログラミングおよびプログラミング言語ワークショップ（PPL2012）, 2012.

[17] 野村芳明, 江本健斗, 松崎公紀, 胡振江, 武市正人. 木スケルトンによる XPath クエリの並列化とその評価. 日本ソフトウェア科学会, コンピュータソフトウェア, Vol.24, No.3, pp. 51–62, 2007.