自己安定平衡探索木構成アルゴリズム

伊藤 瑠美　大下福仁　角川裕次　増澤利光
大阪大学　大学院情報科学研究科

概要　大規模分散システムにおいて，大量の資源を管理するため適切な分散データ構造を用いることが重要である．データを効率的に探索するために有効なデータ構造として平衡探索木が挙げられる．また，システムの拡張性を保証するために故障やデータの挿入・削除など状況の変化に対応する必要がある．動的なシステムの保守には，任意の初期状況から望ましいシステム状況へと自律的に収束することを保証する自己安定性の適用が有効である．そこで，本稿では与えられた任意の探索木から平衡探索木を構成する自己安定アルゴリズムを提案する．

1　はじめに

近年，P2P システムやグリッド・コンピューティングのような大規模**分散システム**の利用が進んでいる．分散システムでは，ノードの離脱や参加，あるいは，故障や復旧などによるシステムの変化が多数発生すると考えられる．分散システムの変化に対する高度な適応性を実現する手法として**自己安定性**[1] が近年注目されている．自己安定性とは，任意の状況からアルゴリズムの実行を開始しても，問題の要求を満たす状況へ到達して安定するという性質である．

大規模分散システムにおいて，大量の資源を管理するため適切な**分散データ構造**を用いることが重要である．分散データ構造は，物理ネットワーク上に論理リンクを用いて**オーバーレイネットワーク**として実現する．仮想的な論理リンクを用いることによって，設計者が物理ネットワークの制約を受けずに，望ましい性質を持つ分散データ構造を構成することができる．データを効率的に探索するために有効なデータ構造として**平衡探索木**が挙げられる．代表的な平衡探索木として **B 木**(図 1) が挙げられる．B 木は各ノードが最大 m 個の子をもつ m 分木であり，さらに，すべての葉ノードのレベルが等しいという平衡条件を満たす．データは葉ノードにのみ格納され，各内部ノードが探索用のラベルを保持する．B 木の各内部ノードは m 個のラベルを保持するので，m を十分に大きく設定することにより，B 木の各ノードを 1 台の計算機で実現する分散データ構造に適すると考えられる．

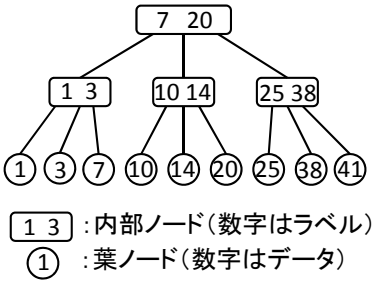

図 1　B 木

B 木に 1 データを挿入・削除すると，ノードのこの数に関する制約と平衡条件を満たすために，B 木の構造変更が必要なことがある．この構造変更は，挿入・削除を行った葉ノードから根ノードまで及ぶことがあり，分散データ構造では，大域的な構造変更を伴うことがある．一方，分散システムのデータ構造では，断続的なデータの挿入・削除や通信遅延などの障害が発生する可能性があり，大域的な構造変更を伴うデータの挿入・削除操作を逐次的に実行するのは非効率的である．そこで，分散 B 木では，データの挿入・削除操作を継続して受付可能とし，一時的には，子の数に関する制約や平衡条件が不成立となることを許して，局所的な構造変更のみで対応することが効率的であると考えられる (図 2)．継続的なデータ追加・削除を局所的に実行すると，平衡条件や次数の制約，データの順序，ラベルの内容など，B 木の厳しい制約が大きく破綻することがある．そこで，本稿では，与えられた任意の初期探索木から分散 B 木を構成する問題を解く自己安定アルゴリズムを提案する (図 3)．本稿で提案するアルゴリズムは，ラベル，データの配置，平衡状態が任意の初期木から局所的な情報のみを用いて自律的に分散 B 木を構成するアルゴリズムである．

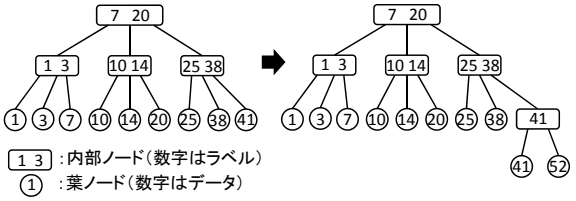

図 2　局所的なデータ挿入

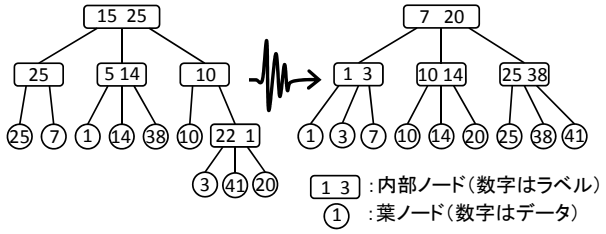

図 3　自己安定 B 木構成アルゴリズム

関連研究

これまで，分散データ構造を構成する様々な自己安定アルゴリズムが提案されてきた [3, 4, 5]．[3] では，ノード数 n に対して収束時間 $O(n)$，空間計算量 $O(\log n)$ で二分探索木を構成する自己安定アルゴリズムを提案している．構成される二分探索木は，初期構造に依存したものとなる．[4] では，与えられた任意の探索木を平衡化する自己安定アルゴリズムを提案している．[4] のアルゴリズムは，平衡化のため探索木内の辺を交換する操作を行う．辺の交換は各ノードが保持するラベルの変更を伴うため，探索の非効率化を引き起こす虞がある．

本稿の構成

本稿の構成は以下のとおりである．まず 2 節において諸定義を行う．次に，3 節において提案アルゴリズムを説明し，最後に 4 節において本研究のまとめと今後の課題を述べる．

2 諸定義

ある定数 m に対して，B 木において，根または葉以外の各ノードは，$\lceil m/2 \rceil$ 以上 m 以下の子をもつとする．また，根は 2 以上 m 以下の子をもつとする．根から各葉ノードまでの距離（深さ）は全て等しいという平衡条件を満たす．データは葉ノードにのみ格納され，各葉ノードは，1 つのデータを格納している．全ての葉ノードがデータが昇順となるように整列している．各内部ノードは複数のラベルを格納している．あるノード u の i 番目のラベルは，u に接続する i 番目と $i+1$ 番目の子を根とする部分木に含まれるデータの境界値である．

分散 B 木は，各ノードをプロセスに対応させることにより実装する．本稿では，分散 B 木のノード数の変化に応じて，プロセスの追加・削除が可能であるとする．各ノードは隣接ノードとのみ通信可能である．通信モデルとして，**局所共有メモリモデル**[2] を用いる．局所共有メモリモデルでは，変数の読み込みによって隣接ノードと通信を行う．各ノードは自身の変数に書き込みが可能で，隣接ノードの変数を読み込み可能である．

3 分散 B 木構成自己安定アルゴリズム

本節では，任意の初期探索木から分散 B 木を構成するアルゴリズムを紹介する．ただし，初期状況において各内部ノードがもつ子の数は m 以下であるとする[1]．

提案アルゴリズムにおいて，各ノード u は以下の変数を保持する．

- $u.child[i]$: u の i 番目の子のノード ID を格納する．
- $u.label[i]$: u の i 番目のラベルを格納する．
- $u.children$: u の子の数を格納する．
- $u.height$: u の高さ (子孫の葉ノードまでの最大距離) を格納する．

ノード u の子の数が $u.children$ のとき，記憶すべき部分木間の境界値は $u.children-1$ 個となる．提案アルゴリズムでは，各ノードを根とする部分木内の最大データを記憶しておく必要がある．そこで本稿では，ノード u を根とする部分木内の最大データを $u.children$ 番目のラベルとして $u.child[u.chiledren]$ に格納しておくこととする．

さらに，ノード u に対して以下の述語を定義する．

- $h\_correct(u)$: u の高さが子の高さと矛盾しないとき $true$ を返す．
- $balanced(u)$: u を根とする部分木が平衡なとき $true$ を返す．
- $ordered(u)$: u を根とする部分木中のデータが昇順に整列されているとき $true$ を返す．
- $labeled(u)$: u のラベルが子のラベルと矛盾しないとき $true$ を返す．

このアルゴリズムは，次の 3 つの自己安定副アルゴリズムから構成される．

1. **データ再配置副アルゴリズム**：探索木に含まれるデータの並びが昇順になるように再配置する．
2. **ラベル調整副アルゴリズム**：各ノードのラベルを正しく設定する．
3. **平衡化副アルゴリズム**：部分木の高さの差を減少させ，平衡探索木を構成する．

提案アルゴリズムでは，これら 3 つの自己安定副アルゴリズムを並行動作させる．各ノード u の動作 (Algorithm1) は次の通りである．

1. ノード u の子の数が m を超えている場合，適切なラベルを親に転送し，自ノードを分割する (図 4)．
2. ノード u の子を根とする各部分木においてデータが昇順に整列され，根が正しいラベルを保持している場合，
   - ノード u を根とする部分木においてデータが昇順に整列されていない場合，データ再配置副アルゴリズム (Algorithm2) を実行する．

[1] 子の数をこのように制限しても，平衡条件を緩めているため，挿入や削除を局所的に実行可能である．

- ノード u のラベルが u の子のラベルと矛盾する場合，ラベル調整副アルゴリズム (Algorithm3) を実行する．

3. ノード u が正しい高さ値を保持し平衡化でなく，かつ正しい高さ値を保持し平衡な v が u の子として存在する場合，平衡化副アルゴリズム (Algorithm4) を実行する．

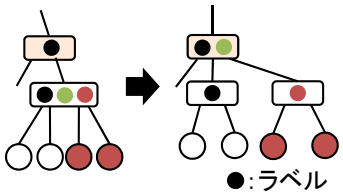

図 4　子が m を超えた場合の処理

3.1　データ再配置副アルゴリズム

データ再配置副アルゴリズムでは，葉ノードに格納されたデータが昇順となるように，データの再配置をおこなう．

各ノード u の動作 (Algorithm2) は次の通りである．各ノードは i 番目と $i+1$ 番目の子を参照する ($1 \leq i \leq u.children - 1$)．$i$ 番目の子の最後のラベル，つまり i 番目の子を根とする部分木の最大値と，$i+1$ 番目の子の最後のラベルを比較し，前者の方が大きい場合に，該当するデータの交換を実行する (図 5)．

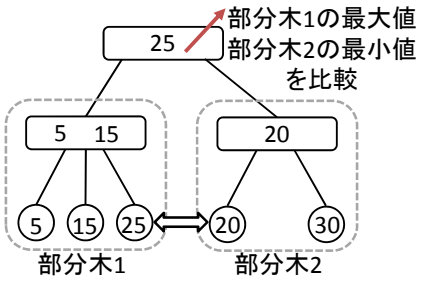

図 5　データ再配置副アルゴリズム

3.2　ラベル調整副アルゴリズム

ラベル調整副アルゴリズムでは，各ノードが正しくラベルを設定する．各ノード u の動作 (Algorithm3) は次の通りである．

各ノードは i 番目の子を参照する ($1 \leq i \leq u.children$)．i 番目の子の最後のラベル，つまり i 番目の子を根とする部分木の最大値を，自身の i 番目のラベルとして設定する (図 6)．

Algorithm 1 分散 B 木構成自己安定アルゴリズム：ノード u の動作

Constants:

m: 子の数の上限

$u.id$: u の ID

Variables:

$u.child$: u の子の集合

$u.child[i]$: u の i 番目の子

$u.label[i]$: u の i 番目のラベル

$u.children$: u の子の数

$u.height$: u の高さ

Predicates:

$h\_correct(u)$: u の高さが子と矛盾ないとき $true$ を返す

$balanced(u)$: u を根とする部分木が平衡なとき $true$ を返す

$ordered(u)$: u を根とする部分木中のデータが整列されているとき $true$ を返す

$labeled(u)$: u のラベルが子と矛盾ないとき $true$ を返す

Function:

repeat forever do

1: **if** $u.children > m$ **then**
2:　ラベルを親に転送し，自ノードを分割;
3: **if** $\forall v \in u.child : (ordered(v) = true$ **and** $labeled(v) = true)$ **then**
4:　**if** $ordered(u) = false$ **then**
5:　　データ再配置副アルゴリズム (Algorithm2) を実行;
6:　**else if** $labeled(u) = false$ **then**
7:　　ラベル調整副アルゴリズム (Algorithm3) を実行;
8: **if** $h_correct(u) = false$ **then**
9:　**if** u は内部ノード **then**
10:　　$u.height \leftarrow 1 + \max v.height$;
11:　**else**
12:　　$u.height \leftarrow 0$;
13: **else if** $balanced(u) = false$ **and** $\exists v \in u.child :$ $(balanced(v) = true$ **and** $h\_correct(v) = true$ **then**
14:　平衡化副アルゴリズム (Algorithm4) を実行;

Algorithm 2 データ再配置副アルゴリズム：ノード u の動作

Function:

1: **for** $i = 1 \ldots u.children - 1$ **do**
2: **if** $(u.child[i]).label[(u.child[i]).children] - (u.child[i+1]).label[1] > 0$ **then**
3: i 番目の部分木の最大データと $i+1$ 番目の部分木の最小データを交換;

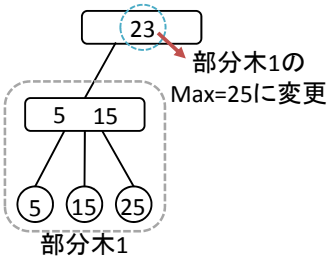

図 6　ラベル調整副アルゴリズム

Algorithm 3 ラベル調整副アルゴリズム：ノード u の動作

Function:

1: **for** $i = 1 \ldots u.children$ **do**
2: $u.label[i] \leftarrow (u.child[i]).label[(u.child[i]).children]$;

3.3　平衡化副アルゴリズム

平衡化副アルゴリズムでは，各ノードが高さの異なる子をもつ場合，高さの差を減少させるように探索木の変形を行う．各ノード u の動作 (Algorithm4) は次の通りである．

ノード u の子のうち，平衡かつ高さの正しいすべてのノード v に対して，ノード v のラベルを u のラベルとして追加する．さらに，v の子を u の子として追加する (図 7)．

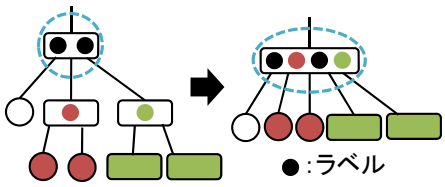

図 7　平衡化副アルゴリズム

Algorithm 4 平衡化副アルゴリズム：ノード u の動作

Function:

1: **for all** $v \in u.child$(ただし，v は $balanced(v) = true$ かつ $v \neq \arg\min_{w \in v.child} w.height$ を満たす) **do**
2: v のラベルを u 自身のラベルとして追加;
3: v の各子を u 自身の子として追加;

4　まとめと今後の課題

本稿では，任意の探索木から平衡探索木である分散 B 木を構成する自己安定アルゴリズムを提案した．提案アルゴリズムでは，データ再配置副アルゴリズム，ラベル調整副アルゴリズム，平衡化副アルゴリズムの 3 つの副アルゴリズムを並行して実行するものである．ラベル調整副アルゴリズム及び平衡化副アルゴリズムでは，初期木の高さを h とすると，収束時間が $O(h)$ となることが予測できる．また，データ再配置副アルゴリズムでは，初期木のデータの逆転数を r とすると，収束時間が $O(hr)$ となると予測できる．収束時間や空間計算量に関する正確な評価は今後の課題とする．

また，提案手法では，データ再配置副アルゴリズムにおいて，データの交換方法を定義していない．葉ノード同士をポインタで接続することで，葉ノード間でのデータ交換を直接可能にし，アルゴリズムの簡略化を図りたい．

参考文献

[1] S. dolev, Self-stabilization. The MIT Press, 2000.
[2] E. W. Dijkstra.: Self-stabilizing systems in spite of distributed control. Commun. ACM 17(11), 643-644 (1974).
[3] D. Bein, A. K. Datta, V. Villain.: Snap-stabilizing optimal binary search tree. Proceedings of the 7th international conference on Self-Stabilizing Systems (2005).
[4] E. Bampas, A. Lamani, F. Petit, M. Valero.: Self-stabilizing balancing algorithm for containment-based trees. Proceedings of the 15th international conference SSS (2013).
[5] S. Bianchi, A. K. Datta, P.Felber, M. Gradinariu.: Stabilizing peer-to-peer spatial filters. In: ICDCS, p. 27. IEEE Computer Society (2007)