データベースシステム　122－6
(2000. 7. 26)

インクリメンタルな更新を伴う全文検索インデックスの分散並列処理方式

吉原 潤
筑波大学大学院修士課程
理工学研究科

加藤 和彦
筑波大学電子・情報工学系
科学技術振興事業団

奈良崎 清彦
筑波大学第三学群
情報学類
(現 日立製作所勤務)

E-mail : {yosiwara, kato, narazaki}@osss.is.tsukuba.ac.jp

あらまし　suffix array はテキストの接尾辞のポインタを接尾辞の辞書順に並べたもので，任意の部分文字列検索を高速に行うことができるが，更新のオーバーヘッドが大きい．本論文では suffix array を効率的に更新する方式として，我々が以前提案したインクリメンタルな更新方式を分散並列化をした方式を提案する．この方式では suffix array に含まれる接尾辞を辞書順のある範囲で分割し，各ノードに担当区間を割り当てる．繰り返される更新に伴い各ノードの担当区間のサイズの不均衡が生じるため，動的に担当区間の変更を行ない更新処理の負荷を均等化する．また，単純に均等なサイズに分割して連続した区間をノードに割り当てた場合に検索要求の分布に偏りが生じることを示し，検索要求の偏りを軽減する分割方法を提案した．

キーワード　suffix array, 分散並列処理, 全文検索

Distributed Parallel Processing Scheme of a Full-Text Index Structure with Incremental Updating

Jun Yoshiwara
Master's Program in Science and Engineering
University of Tsukuba

Kazuhiko Kato
Institute of Information Science and Electronics
University of Tsukuba
Japan Science and Technology Corporation

Kiyohiko Narazaki
College of Information Sciences
Third Cluster of Colleges
University of Tsukuba

Abstract

A suffix array is a full-text index data structure which is efficient for retrieving any substring of text, but requires a lot of overhead for updating it. In this paper, we propose an efficient updating scheme of suffix arrays. In this scheme, a suffix array is split into some sections and each section is assigned to a node. When updating, the incremental updating scheme which we already proposed runs in parallel on each node. To balance the sizes of sections after repeated updating, boundaries of sections are changed dynamically. Furthermore we propose the spliting scheme of suffix arrays to balance the retrieval prosessing load.

key words　suffix array, distributed parallel processing, full-text search

1 はじめに

ここ数年，インターネット環境の急速な普及に伴い WWW が一般に広く利用されるようになり，WWW で提供される膨大な量の情報の中から求める有用な情報を見つけ出す WWW 検索エンジンが情報収集のための重要な手段となっている [4].

大規模テキストに対する文字列の検索は，あらかじめ単語として切り出された単語を検索の対象とする方法と，任意の部分文字列を対象として検索する方法とに大別される．前者の処理を高速に行うための方法として，転置ファイルやシグネチャファイル等のインデックス構造を用いた方法がよく知られている．任意の部分文字列を検索する方法としては，suffix array[2], suffix tree[3], SB-tree [1] などのインデックス構造を用いた方法が知られている．suffix array は，1990 年に Manber と Myers によって提案されたインデックス構造で，テキストの接尾辞を辞書順にソートして，その接尾辞へのポインタを並べた配列である．suffix array は他の 2 つよりもコンパクトな構造であり，また，二分探索によって高速に任意の部分文字列の検索ができるため，テキストが巨大になる場合にも有効である．

WWW 検索エンジンではインデックスを作る対象となるテキストが頻繁に更新されるので，それに合わせてインデックスも更新し，常にできるだけ新しい情報を検索結果として提供する事が要求される．suffix array は，検索漏れがない全文テキスト検索技術であり，高速な検索が可能で，コンパクトなインデックス構造を持つが，静的な構造であるため，テキストが頻繁に更新される場合それに伴う suffix array の更新にかかるオーバーヘッドが非常に大きい．我々は既に suffix array のインクリメンタルな更新方式 [5] を提案した．この方式では，単一の巨大な suffix array を更新していくのではなく，差分のテキストをもとに差分の suffix array を作っていき，検索はそれら複数の suffix array すべてに対して行う．ある程度の個数の差分 suffix array が作られたらそれらをマージして単一の suffix array を再構成する．この方式では，差分の suffix array を作るのは高速だが，単一の suffix array を再構成する処理のコストが大きい．

そこで本論文では，これらの処理を分散並列化し，より効率的に suffix array の更新処理を行う方式を提案する．この方式では，suffix array を分割して複数のサイトに配置し，各サイトで並列して更新処理を行なう．その際，各サイトでの負荷が均等になるように suffix array のサイズが均等になるように領域に分割するが，そのままでは各領域のもつ接尾辞の質に偏りが生じ，検索要求の分布に偏りが生じる．そこで検索処理の負荷も均等化するため，それを改善した分割方法を示す．

2 基本概念

本章では，以下に続く章を理解するために必要な基本概念として，suffix array と，文献 [5] で提案したインクリメンタルな更新方式について説明する．

2.1 Suffix array

テキスト $T[1,n]$ に対して，$T[i,n], T[1,j]$ をそれぞれ T の接尾辞，接頭辞という．入力テキスト $T[1,n]$ の suffix array $A[1,n]$ とは，T の接尾辞 $s_i = T[i,n]$ へのポインタ i $(i = 1,\ldots,n)$ を，それが指す接尾辞の辞書順に並べた配列である．よって $T[A[i],n]$ は辞書順で i 番目になる接尾辞となる．A を用いて T を参照し配列 $T[A[1],n],\ldots,T[A[n],n]$ を考えると，これは辞書順に並んだ T の接尾辞の配列になっている．suffix array の例を図 1 に示す．

2.2 インクリメンタルな更新方式

文献 [5] で，我々は suffix array のインクリメンタルな更新方式を提案した．この方式の概要を示す．

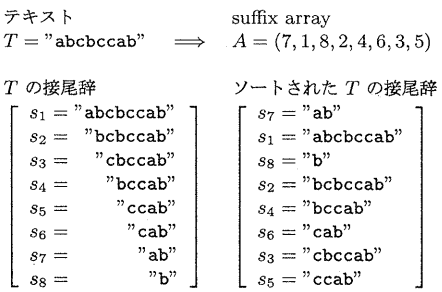

図 1: suffix array の例

この方式では，繰り返されるテキストの更新があるたびに単一の suffix array を直接に更新する訳ではない．複数の suffix array を作成し，検索ではこれら全てに対して検索しその結果をマージする．

まず，テキストの初期集合 Δ_0 を連結し一つの文字列とし，これに対し suffix array を作成する．これを主インデックス I_0 とする．その後，Web ロボットがテキストを収集してくるなどして，インデックスを作成する対象となるテキストが追加された場合，元のテキストおよび suffix array はそのままとし，追加された差分のテキスト集合 Δ_1 から，それを連結した文字列とその suffix array を構成する．これを差分インデックスと呼び，I_1 とする．更にその後のテキスト集合 Δ_2 の追加に対しては，同様にテキストを連結し suffix array を作成する．これを先に作ってある差分インデックス I_1 とマージする．何度かこのようなマージを繰り返し，マージ処理にかかるコストがある程度大きくなったら，次の差分インデックス I_2 を作成する．以後同様に，追加された差分テキストを最後に作られた差分インデックスにマージしていく (図 2)．

テキストが削除された場合，基本的には削除をしない．削除すべきテキストが最新の差分インデックスに含まれる場合は実際に削除し，それ以外の場合は削除すべきテキストの区間だけを記録しておいて後でまとめて削除する．そうすると，古い内容を持ったテキストと新しい内容を持ったテキストが混在してしまう．そこで有効インデックス番号というものを用いる．これは，各テキストについて，そのテキストの最新の内容がどの suffix array に含まれているかを記録するもので，テキストの追加の際に更新される．これを用いて，検索処理の際に古くなって無効な情報を取り除く．

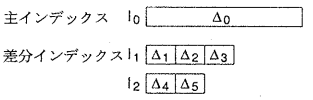

図 2: インクリメンタルな更新方式

更新を繰り返し差分インデックスの個数が増加するにつれて，検索コストが大きくなってしまう．そこで，差分インデックスの個数がある値を越えたら，全体をマージして単一の suffix array を再構成する．このとき，削除されたことになっているテキストを取り除く．

十分大きな量のテキストに対するインデックスの場合，保持している全テキストに対し更新された差分のテキストは十分少ないため，suffix array 全体を更新するよりも別個に suffix array を作成する方がコストが小さく，また，差分インデックスは suffix array 全体に比べ小さいのでマージ処理のコストもあまり大きくならない．

3 更新処理の分散並列化

文献 [5] のインクリメンタルな更新方式では，単一の計算機上で処理を行なっていた．この章では，インクリメンタルな更新方式をを分散並列化することでより効率的に更新を行なう方式を提案する．

3.1 Suffix array の分割

Suffix array の更新処理を分散，並列化するため，suffix array を分割し，複数のサイトで更新

処理を行う．

suffix array $A[1,n]$ は辞書順に並んだ接尾辞へのポインタの配列であるため，辞書順のある範囲ごとの領域に分割できる．そこで，$d_i < d_{i+1}$ を満たす文字列 $d_1,\ldots,d_{m-1}$ によって m 個の区間 $R_1,\ldots,R_m$ に分割する．ここで $R_i = A[k_{i-1}+1,\ldots,k_i]$ について，k_i は $T[A[r_i],n] < d_i$ を満たす最大の値である．また $k_0 = 0, k_m = n$ である．この各区間の境界となる文字列 d_i を分割文字列と呼ぶ．

分割した suffix array の領域を保持する m 個のスレーブノードと，それらをまとめて管理する一つのマスターノードがある．マスターノード M では分割文字列情報を保持する．i 番目のスレーブノード S_i は区間 R_i に含まれる接尾辞だけからなる suffix array を持てばよい．一方で，suffix array は元になっているテキストと対になってインデックスの役割を果たすため，テキストについては全てのスレーブノードでテキスト全体のコピーを保持しなければならない．しかし，テキストのサイズは suffix array の更新処理のコストにほとんど影響しないので問題は無い．

以下に，差分のテキスト $T[1,n]$ を追加する際の処理の手順を示す．この処理の様子を図 3 に示す．

マスターノード M において:

1. 差分のテキスト集合を連結し単一のテキスト $T[1,n]$ を作る．テキスト T に対する suffix array $A[1,n]$ を作る．
2. $A[1,n]$ を二分探索し分割文字列 $d_1,\ldots,d_{m-1}$ の位置を見つけ，A を区間 $R_1,\ldots,R_m$ に分割する．
3. スレーブノード S_i $(i=1,\ldots,m)$ にテキスト T 全体と suffix array の担当区間 R_i を渡す．

スレーブノード S_i において:

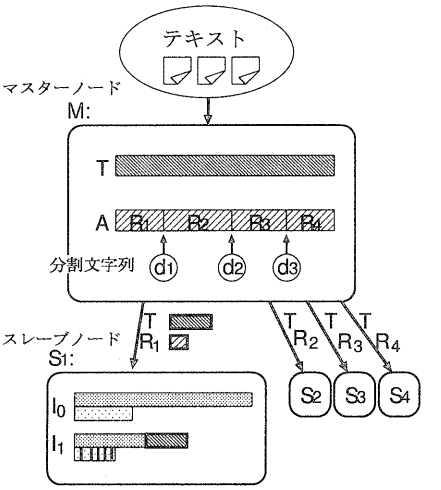

図 3: suffix array の更新処理の分散並列化

1. マスターノードから差分のテキスト全体 T と自分の担当区間である suffix array の区間 R_i を受け取る．
2. インクリメンタルな更新方式に従い，次のどちらかの処理をする．どちらを行なうかはパラメータの設定に因る．
   - T と R_i を新規の差分インデックスとする．
   - T と R_i を既存の最新インデックスとマージする．

次に，テキスト $T[1,n]$ を削除する際の処理の手順を示す．

1. マスターノードが，全てのスレーブノードに対し削除されたテキストを通知する．
2. 各スレーブノードは，通知された削除すべきテキストが最新の差分インデックスに含まれていればその削除処理を行ない，それ以外の場合は，有効インデックス番号の更新と削除すべき位置の記録を行なう．

suffix array を分割して複数のスレーブノードに配置したことにより，検索処理も複数のスレーブノードで並列して行なうことができる．検索要求はまずマスターノードが受け付ける．マスターノードは分割文字列を用いてクエリ文字列がどのスレーブノードの担当区間に含まれているかを調べ，そのスレーブノードに検索要求を伝える．そのスレーブノードで検索処理を行ない，マスターノードに結果が返される．クエリ文字列が分割文字列の部分文字列となっている場合，そのクエリ文字列は二つのスレーブノードに跨って存在することになる．この場合，両方のスレーブノードに検索要求を出し，得られた結果をマージする．

3.2 動的な担当区間の変更

更新処理のコストは，処理する suffix array のサイズに依存する．したがって各スレーブノードでの負荷が均等に成るようにするためには，担当区間 $R_1, \ldots, R_m$ のサイズが均等になるように分割文字列を設定する必要がある．

最初に suffix array を作ったときには，それを m 等分する位置にある接尾辞を分割文字列とすれば良い．しかし，その後に追加，削除されるテキストに含まれる接尾辞は，最初に分割された各領域に対して同じ割合で出現する訳ではない．よって更新処理を繰り返すうちに，各区間のサイズに偏りが生じてくる．そこで，偏りがある程度大きくなってきたら，各区間のサイズを調整し負荷を均等化する処理が必要となる．

suffix array は辞書順に並んだ接尾辞へのポインタの配列なので，複数の領域に分割したり，ある領域同士を単純に連結したりすることができる．そこで，各区間のサイズに偏りが生じたら，以下のような手順で各スレーブノードの担当区間を変更する．ここで，これまでの分割文字列を $d_1, \ldots, d_{m-1}$，区間を $R_1, \ldots, R_m$，またサイズ均等化後の分割文字列を $d'_1, \ldots, d'_{m-1}$，区間を $R'_1, \ldots, R'_m$ とする．この処理の様子を図 4 に示す．

1. マスターノードがスレーブノード S_i に，現在保持している担当区間の suffix array R_i のサイズ $|R_i|$ を問い合わせる．

2. マスターノードが，サイズ均等化後の各区間のサイズ s を以下のようにして求める．ここで，σ_i は区間 R_1 から R_i までのサイズの合計値である．

$$s = \frac{1}{m}\sigma_m = \frac{1}{m}\sum_{j=1}^{m}|R_j|$$

3. $R_1, \ldots, R_m$ を連結して作られる suffix array を $A[1,n]$ とすると，区間 R_i は $A[\sigma_{i-1}+1, \sigma_i]$ にあたる．また，サイズ均等化後の区間 R'_j は $A[(j-1)s+1, js]$ にあたる．

   マスターノードは，各スレーブノード S_i に対し，新たな区間のサイズ s と，$A[1,n]$ 内における R_i の先頭位置 $h_i(=\sigma_{i-1}+1)$ を通知する．

4. $h_i \leq ks < h_i + s$ のとき，R'_k と R'_{k+1} の境界 ks が区間 R_i 内にあることになる．スレーブノード S_i は，マスターノードから通知された情報 s, h_i を用いて，自分が現在保持している区間 R_i が区間更新後にどこで分割されそれらがどのノードの担当になるかを計算する．

5. $x := 0, y := 0$ とする．

6. $x < m$ ならば，マスターノードが，S_y から順にスレーブノードに対して区間 R'_x に含まれる接尾辞を持っているかを問い合わせていき，持っていないと答えたスレーブノード S_z を見つける．S_z に，R_x と R_{x+1} の境界にある新たな分割文字列 d'_x を問い合わせる．また $x = m$ ならば $z := m$ である．

7. マスターノードは，スレーブノード S_i $(i = y, \ldots z)$ に対して，区間 R'_x に含まれる接尾辞を S_x に渡すように通知し，S_x に対して

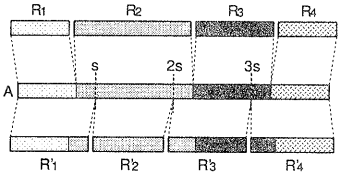

図 4: 動的な担当区間変更

は S_i $(i = y, y+1, \ldots, z)$ から suffix array を受け取るように通知する.

8. スレーブノード S_i $(i = y, \ldots, z)$ が S_x にその担当領域となる suffix array を渡す.

   スレーブノード S_x では, 他のスレーブノードから受け取った suffix array を連結して新たな担当区間 R_x を形成する.

9. $x = m$ ならば終了. そうでなければ $x := x+1, y := z$ として 5 に戻る.

以後, 新たに設定した分割文字列を用いて各スレーブノードに更新すべきデータや検索要求を分配する.

4 検索コストの負荷均等化

3 章では, 更新処理のコストの均等化のみを考え, suffix array を単純に均等なサイズに分割した. しかし, suffix array は文字コード順に接尾辞が並んでいるため, 単純に前から分割していったのでは, 各サイトの担当する区間が含んでいる接尾辞の質に大きな差が生じる. 例えば, ある区間ではあまり検索に用いられない記号類で始まる接尾辞のみからなるのに対し, ある区間では全て漢字から始まる接尾辞からなるということが生じる. このような場合, 各サイトに割り振られる検索要求の数に偏りが生じる. そこで, 更新処理のコストだけでなく検索処理のコストも同時に分散できるような方法について考える.

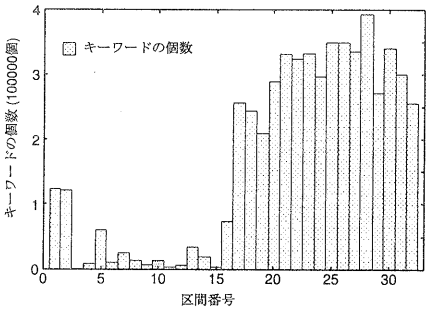

図 5: キーワードの分布

4.1 検索要求の偏り

検索要求の偏りを調べるため, 以下のような実験を行なった.

まず, 毎日新聞の 1998 年度の記事 (364 日分, 約 220 MByte) を元に suffix array を構成し, それを 32 個の領域に分割した. これに対し, 同じ新聞のテキストから切り出したキーワード約 540 万語がそれぞれどの区間に含まれるかを計測した. その結果を図 5 に示す.

この図で, 3 番目から 15 番目の区間は極端にキーワードの分布が少ない. この分布が少ない領域は, 平仮名からはじまる接尾辞が集まっている. 一方, 後半の片仮名, 漢字からはじまる接尾辞をもつ区間にほとんどのキーワードが集中しているのが分かる. つまり, 平仮名は元の新聞記事テキストに含まれる文字の半分近くを占めるにも関わらず, キーワードの先頭の文字としてはあまり出現しなかったということである. ここで用いたキーワードはもとのテキストに含まれているものを抜き出してあるだけなので, 実際に検索に使われるキーワードとは出現頻度の偏りが異なるかも知れないが, 一般的に検索で使われるキーワードの場合も同様に平仮名のものの割合は低いと考えられる.

このように, suffix array を単純に前から分割していった場合, ある区間ではあまり検索に用

いられない接尾辞の割合が高いのに対し別の区間では検索に用いられる接尾辞が多く含まれるということが生じ，検索処理の負荷の偏りが著しくなることが分かる．

そこで検索処理の負荷を均等に分散するように suffix array を分割する方法を考える．すなわち，検索要求に与えられた文字列の十分大きなサンプルを集め，これらをソートして m 等分した境界にある文字列を分割文字列とする．しかしこの方法では，各スレーブノードが担当する suffix array のサイズを考慮していないため，検索要求のくる頻度が少ない領域では大きな suffix array を持つことになり，更新処理の負荷に偏りが生じることになる．

そこで更新処理のコストを均等に分散させたまま検索要求の偏りを軽減するためような suffix array の分割方法を考える．各スレーブノードが担当する区間は，必ずしも元の suffix array で一つの連続した領域である必要は無い．そこで，一つのスレーブノードが担当する区間を複数のブロックで構成する．まず，suffix array を先頭の 1 文字の種類によって分割する．すなわち，先頭の文字が英数字，平仮名，片仮名，漢字である領域に分割する．次に，各領域をサイズが m 等分になるように分割する．このように 2 段階で分割した後，各領域での i 番目の区間を連結し，これを i 番目の担当区間 R_i とする．このような分割方法で動的に区間変更をする場合，平仮名，片仮名などの各領域ごとに 3.2 節の方法で新たな分割位置を求め，それぞれを連結して新しい区間を決める．

4.2 検索の偏りに対する動的な負荷均等化

suffix array を分割した各領域のサイズの比は繰り返される更新に伴い変化していく．同様に，検索要求の分布も常に一定ではなく，時間により変化していくと考えられる．よって検索要求の偏りの変化にも対応できる方がより望ましいと考えられるが，前節の分割方法ではサイズの均等化を優先したためそれはできない．

検索要求の偏りに対応するためには，検索要求の履歴を持ち，検索要求の分布を調べる必要がある．例えば B^+-tree に検索で使われたキーワードを記録していき，ある件数を越えたら古いものから捨てていくようにすることで最近の検索要求の分布を得ることができる．

更新処理と検索処理の負荷均等化を両立するような suffix array の分割方法として，suffix array を細かい単位のブロックに分割し，以下の条件を可能な限り満たすようにブロックを各スレーブノードに分配する制約最適化問題を解くという方法が考えられる．

- 各スレーブノードの担当する区間のサイズの大きさは等しい
- 各スレーブノードの担当する区間の検索要求の分布の比率は等しい
- 各スレーブノード間でのデータの移動を最小とする．

5 おわりに

本論文では，suffix array を効率的に更新する方式として，我々が以前提案したインクリメンタルな更新方式を分散並列化をした方式を提案した．この方式では suffix array に含まれる接尾辞を辞書順のある範囲で分割し，各スレーブノードに担当区間を割り当てる．繰り返される更新に伴い各スレーブノードの担当区間のサイズの不均衡が生じるため，動的に担当区間の変更を行ない更新処理の負荷を均等化する．また，単純に均等なサイズに分割して連続した区間をスレーブノードに割り当てた場合，検索要求の分布に偏りが生じることを示し，検索要求の偏りを軽減する分割方法を提案した．

今後の課題としては，提案方式を実装し更新処理および検索処理の負荷を分散させることを確かめること，および，実際の Web 検索に用いられるキーワードを収集してその偏りを調べることなどを予定している．

参考文献

[1] Paolo Ferragina and Roberto Grossi. The string b-tree: A new data structure for string search in external memory and its applications. In *JACM 46(2)*, pp. 236–280, 1999.

[2] U. Manber and E.W. Myers. Suffix arrays: A new method for on-line string searches. In *Proceedings of the 1stAnnual ACM-SIAM Symposium on Discrete Algorithms*, pp. 319–327, San Francisco, January 1990.

[3] E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, Vol. 23, No. 2, pp. 262–272, 1976.

[4] W. Sonnenreich and T. Macinta. *Guide to Search Engines.* John Wiley & Sons, Inc., 1998.

[5] 吉原潤, 加藤和彦. Www 検索エンジンのためのインクリメンタルな全文検索インデックス更新方式. 情報処理学会論文誌：データベース, Vol. 40, No. SIG8(TOD4), pp. 112–125, 1999.